ETSI TS 102 114 V1.6.1 (2019-08)

DTS Coherent Acoustics;
Core and Extensions
with Additional Profiles

EBU

Reference
RTS/JTC-DTS-R5

Keywords
audio, broadcast, codec, DVB

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

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

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Contents

List of tables

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (https://ipr.etsi.org/).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

Foreword

This Technical Specification (TS) has been produced by Joint Technical Committee (JTC) Broadcast of the European Broadcasting Union (EBU), Comité Européen de Normalisation ELECtrotechnique (CENELEC) and the European Telecommunications Standards Institute (ETSI).

NOTE: The EBU/ETSI JTC Broadcast was established in 1990 to co-ordinate the drafting of standards in the specific field of broadcasting and related fields. Since 1995 the JTC Broadcast 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
CH-1218 GRAND SACONNEX (Geneva)
Switzerland
Tel: +41 22 717 21 11
Fax: +41 22 717 24 81

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

1 Scope

The present document describes the key components of the DTS Coherent Acoustics technology which is known in the market as DTS-HD™. Prior editions of the present document added Annexes describing stream encapsulations of the bitstreams defined herein for MPEG-2 Transport Stream (based on ISO/IEC 13818-1) and ISO Based Media Files (using ISO/IEC 14496-12 [5]). This edition has been extended with two new Annexes that describe particular methods of network distribution of the defined bitstreams using MPEG-DASH (ISO/IEC 23009-1 [3]) and MPEG-CMAF (ISO/IEC 23000-19 [4]).

2 References

2.1 Normative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at https://docbox.etsi.org/Reference/.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are necessary for the application of the present document.

[1] ETSI EN 300 468: "Digital Video Broadcasting (DVB); Specification for Service Information (SI) in DVB systems".

[2] ETSI TS 101 154: "Digital Video Broadcasting (DVB); Specification for the use of Video and Audio Coding in Broadcast and Broadband Applications".

[3] ISO/IEC 23009-1 (2014): 'Information technology - Dynamic adaptive streaming over HTTP (DASH) -Part 1: Media presentation description and segment formats'.

NOTE: Available at International Standards Organization, www.iso.ch; International Electrotechnical Commission, www.iec.ch.

[4] ISO/IEC 23000-19 (2018): "Information technology - Multimedia application format (MPEG-A) -- Part 19: Common media application format (CMAF) for segmented media'.

NOTE: Available at International Standards Organization, www.iso.ch; International Electrotechnical Commission, www.iec.ch.

[5] ISO/IEC 14496-12: "Information technology - Coding of audio-visual objects - Part 12: ISO Base Media File Format".

NOTE: Available at International Standards Organization, www.iso.ch; International Electrotechnical Commission, www.iec.ch.

[6] Recommendation ITU-T H.222.0/ISO/IEC 13818-1: "Information Technology - Generic coding of moving pictures and associated audio information: Systems".

NOTE: Available at International Standards Organization, www.iso.ch; International Electrotechnical Commission, www.iec.ch.

[7] Void.

[8] Void.

[9] ISO 639-2:1998: "Codes for the representation of names of languages - Part 2: Alpha-3 code".

NOTE: Available at International Standards Organization, www.iso.ch; International Electrotechnical Commission, www.iec.ch.

[10] ISO/IEC 8859-1 (1998): "Information technology - 8-bit single-byte coded graphic character sets - Part 1: Latin alphabet No. 1".

NOTE: Available at International Standards Organization, www.iso.ch; International Electrotechnical Commission, www.iec.ch.

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] DTS Document #9302J81100: "DTS-HD® PBR API Library Description".

NOTE: Available from DTS, Inc., www.dts.com.

3 Definition of terms, symbols, abbreviations and document conventions

3.1 Terms

For the purposes of the present document, the following terms apply:

audio frame: complete logical access unit of an audio stream that corresponds to a defined number of decodable PCM audio samples for a given time segment of the audio presentation

audio stream: sequence of synchronized audio frames

bit(*n*): pseudo type where the parameter ***n*** represents consecutive bits in the stream

NOTE: Padding is never assumed where this is used. All stream parameters described using *bit()* are unsigned and MSB first aligned in the stream.

ByteAlign(): pseudo function to pad to the end of the current byte with from 0 to 7 bits

boolean: value which resolves to either a logical 1 or 0

core substream: audio stream component that adheres to the original DTS Coherent Acoustics definition

dependent substream: specific type of extension substream that is associated with a core substream

DTS Core Audio Stream: carries the coding parameters of up to 5.1 channels of the original LPCM audio at up to 24 bits per sample with the sampling frequency of up to 48 kHz

DTS Extended Audio Stream: channel or frequency extension appended to the core audio component in the core substream

DTS XCH Stream: DTS extended audio stream that carries the coding parameters for 1 additional channel, located in the centre rear position, of original LPCM audio at up to 24 bits per sample with the sampling frequency of up to 48 kHz

DTS X96 Stream: DTS extended audio stream that enables encoding of original LPCM audio at up to 24 bits per sample with the sampling frequency of up to 96 kHz

NOTE: The stream carries the coding parameters used for the representation of all remaining audio components that are present in the original LPCM audio and are not represented in the core audio stream.

duration: time represented by one decoded audio frame, may be represented in audio samples per channel at a specific audio sampling frequency or in seconds

extension: audio stream component providing a specific enhancement or coding profile

extension substream: audio stream component that adheres to the definitions described in clause 7

ExtractBits (*n*): pseudo-function that extracts next ***n*** consecutive bits from the stream

false: Boolean logic value = 0

LBR: DTS-HD extension used to implement the low bit rate coding profile

Linear Pulse Code Modulated (LPCM): sequence of digital audio samples

main audio: default audio presentation

PES payload: portion of the PES packet following the PES header

primary audio channels: audio channels encoded in the DTS core

primary audio presentation: synonymous with main audio

QMF bank: specific filtering structure that provides the means of translating the time domain signal into the multiple subband domain signals

secondary audio: auxiliary or supplemental program

SPDIF: generically referring to S/PDIF or TOSLINK serial audio interfaces

substream: sequence of synchronized frames comprising one of the logical components of the audio stream

true: Boolean logic value = 1

uimsbf: unsigned integer most significant bit first

vector quantization: joint quantization of a block of signal samples or a block of signal parameters

X96: extension that contains the spectrum beyond 24 kHz to compliment a specific set of coded audio stored at a sampling frequency of 48 kHz, permitting source material originally sampled at 96 kHz to be played back on both 4 kHz and 96 kHz capable systems

XBR: extension containing resolution enhancements to the audio elements stored in the core substream

XCH: extension that adds a centre surround channel

XLL: lossless audio coding extension

XXCH: extension that can add up to 32 channels at arbitrary locations to a DTS core (which is limited to up to 5.1 channels) in either the core substream, or in an extension substream

3.2 Symbols

Void.

3.3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

ADPCM	Adaptive Differential Pulse Code Modulation
ASPF	Audio Sync Word Insertion Flag
CA	Coherent Acoustics
CBR	Constant Bit Rate
CRC	Cycle Redundancy Check
DNG	Dialog Normalization Gain
DRC	Dynamic Range Control
DSP	Digital Signal Processor
ES	Extended Surround
FIR	Finite Impulse Response
HCRC	Header CRC Check Bytes
HFLAG	Predictor History Flag Switch
LAR	Linear Area Ratios (as defined in clause 8.5.4 prior to usage)
LFE	Low Frequency Effects
LFF	Low Frequency Effects Flag
LPCM	Linear Pulse Code Modulation
LR	Left Right
LSB	Least Significant Bit
MDCT	Modified Discrete Cosine Transform
MIPS	Million Instructions per Second
MSB	Most Significant Bit
PCM	Pulse Code Modulation
PES	Packetized Elementary Stream
PID	Package Identifier
PMT	Program Map Table
QMF	Quadrature Mirror Filter
VBR	Variable Bit Rate
VQ	Vector-Quantized

3.4 Document Conventions

A number of conventions are applied throughout the present document:

- In parameter descriptions, most significant byte and most significant bit first, (big endian), convention is utilized unless otherwise specified.
- The existence of many bit fields is determined by conditions of prior bits in the stream. As such, in many cases the bit stream elements are described using standard 'C' conventions, with a pseudo function **`ExtractBits()`** representing the bit field of interest for that description.
- Bit field descriptions are described in presentation order.
- In many cases, variable names are assigned to fields as they are being described. In some cases, the variable may be modified during definition, such as:

```
nuFieldSize = ExtractBits(4)+1
```

4 Summary

4.1 Overview

The DTS Coherent Acoustics coding system is designed to deliver advanced features while supporting compatibility to a large installed base of decoders. The addition of the extension substream to the coding system permits a great deal of flexibility in defining new coding profiles and major feature enhancements. By adding a low bit rate profile and a lossless coding extension, the augmented system is able to address a broader range of applications. Metadata and new extensions increase versatility, such as embedded replacement channels, dynamic mixing coefficients and alternate channel maps. Instantiations of this new coding system exist in the market today and are known as DTS-HD™, including DTS-HD High Resolution Audio™ and DTS-HD Master Audio™.

Due to the popularity of the 5.1 channel sound tracks in the movie industry and in the multi-channel home audio market, DTS Coherent Acoustics is delivered in the form of a core audio (for the 5.1 channels) plus optional extended audio (for the rest of the DTS Coherent Acoustics). The 5.1 channel audio consists of up to five primary audio channels with frequencies lower than 24 kHz plus a possible Low Frequency Effect (LFE) channel (the 0.1 channel). This implies that the frequency components higher than 24 kHz for the five primary audio channels and all frequency components of the remaining two channels are carried in the extended audio.

- Core Audio:
  - Up to 5 primary audio channels (frequency components below 24 kHz).
  - Up to 1 low frequency effect (LFE) channel.
  - Optional information such as time stamps and user information.
  - One channel extension ((XCH or XXCH) up to 7.1 channels) or a frequency extension (X96) for up to 5.1 channel.
- Extended Audio:
  - Up to 32 additional full bandwidth channels (frequency components below 24 kHz).
  - Frequency components above 24 kHz for the primary and extended audio channels.
  - Resolution enhancement for primary channels in the core audio, including up to bit exact replication of the audio source material (lossless compression).
  - Hierarchically embedded downmix capability.
- Lossless coding algorithm:
  - Capable of extending the original DTS core to render a bit-exact reproduction of the studio master in a bit-stream with inherent legacy compatibility.
  - Capable of operating in a stand-alone mode for maximum efficiency when legacy compatibility is not required.

With the core + extension structure, the core is always coded without a reference to the extension, allowing the core substream to be decoded independently. A sophisticated decoder, however, can first decode the 5.1 core audio bits and then proceed to decode the extension substream to enhance sonic accuracy (up to and including lossless), add channels and extend the audio sampling frequency.

When bit efficiency is paramount and legacy compatibility is not an issue, the use of the extension substream with the LBR extension provides a flexible and robust solution. Some of the advantages of the low bit rate profile are:

- Optimized for bit rate sensitive broadcasting and internet streaming.
- 5.1 channels of broadcast quality audio in as low as 192 Kbits/s.
- Up to 7.1 channel capability.

- Same rich metadata feature set as is available for other DTS formats.

4.2 Organization of the present document

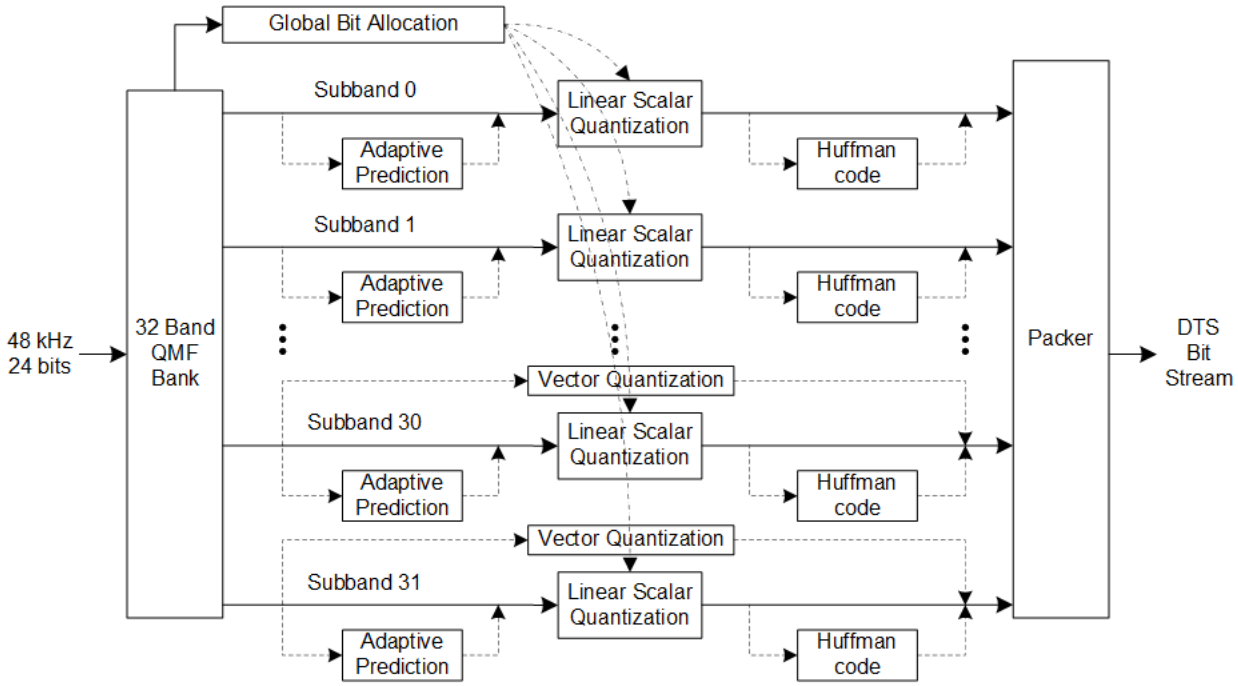
The present document will describe the fundamental elements of this coding system, including:

- Clause 5: Core Audio - Bit stream syntax of the DTS core header and coding of DTS Coherent Acoustics (a.k.a. core, or core audio) and DTS core metadata extensions.
- Clause 6: Core Extensions - Bit-stream syntax of DTS core extensions used to enhance channel count and sampling frequency.
- Clause 7: Extension Substream - Bit stream syntax of the header and metadata components.
- Clause 8: Lossless Coding Extension - Bit stream syntax and coding.
- Clause 9: LBR Coding Extension - Bit stream syntax and coding.
- In the annexes:
  - Supplemental algorithms useful in decoding the DTS bitstream.
  - Look-up tables.
  - Information necessary to embed the DTS core and extension substream components into ISO media files (in accordance with ISO/IEC 14496-12 [5]).
  - Guidance on embedding the DTS core and extension substream components into MPEG-2 broadcast streams, referring to ETSI TS 101 154 [2] and ETSI EN 300 468 [1] and in accordance with ISO/IEC 13818-1 [6].
  - Transmission of the described bitstream in a digital network using MPEG standards.

5 Core Audio

5.1 Introduction

The DTS core encoder delivers 5.1 channel audio at 24 bits per sample with a sampling frequency of up to 48 kHz. As shown in Figure 5-1, the audio samples of a primary channel are split and decimated by a 32-band QMF bank into 32 subbands. The samples of each subband goes through an adaptive prediction process to check if the resultant prediction gain is large enough to justify the overhead of transferring the coefficients of prediction filter. The prediction gain is obtained by comparing the variance of the prediction residual to that of the subband samples. If the prediction gain is big enough, the prediction residual is quantified using mid-tread scalar quantization and the prediction coefficients are Vector-Quantized (VQ). Otherwise, the subband samples themselves are quantized using mid-tread scalar quantization. In the case of low bit rate applications, the scalar quantization indexes of the residual or subband samples are further encoded using Huffman code. When the bit rate is low, Vector Quantization (VQ) may also be used to quantize samples of the high-frequency subbands for which the adaptive prediction is disabled. In very low bit rate applications, joint intensity coding and sum/difference coding may be employed to further improve audio quality. The optional LFE channel is compressed by: low-pass filtering, decimation and mid-tread scalar quantization.



NOTE: The dotted lines indicate optional operations and dash dot lines bit allocation control.

Figure 5-1: Compression of a primary audio channel

5.2 Frame structure and decoding procedure

DTS bit stream is a sequence of synchronized frames, each consisting of the following fields (see Figure 5-2). A pseudo code overview of the main function calls is listed in clause C.3:

- **Synchronization Word:** Synchronize the decoder to the bit stream.
- **Frame Header:** Carries information about frame construction, encoder configuration, audio data arrangement and various operational features. See clause 5.4 for details of the Frame Header.
- **Subframes:** Carries core audio data for the 5.1 channels. Each frame may have up to 16 subframes. See clause 5.5 for details of the primary audio coding header routines.
- **Optional Information:** Carries auxiliary data such as time code, which is not intrinsic to the operation of the decoder but may be used for post processing routines. Some optional metadata has been defined in clause 5.8.
- **Extended Audio:** Carries possible extended channels and frequency bands of the primary audio channels as well as all frequency components of channels beyond 5.1. These extensions are described in clause 6.

Each subframe contains data for audio samples of the 5.1 channels covering a time duration of up to that of the subband analysis window and can be decoded entirely without reference to any other subframes. A subframe consists of the following fields (see Figure 5-3):

- **Side Information:** Relays information about how to decode the 5.1 channel audio data. Information for joint intensity coding is also included here.
- **High Frequency VQ:** A small number of high frequency subbands of the primary channels may be encoded using VQ. In this case, the samples of each of those subbands within the subframe are encoded as a single VQ address.
- **Low Frequency Effect Channel:** The decimated samples of the LFE channel are carried as 8-bit words.
- **Subsubframes:** All subbands, except those high-frequency VQ encoded ones, are encoded here in up to 4 subsubframes.

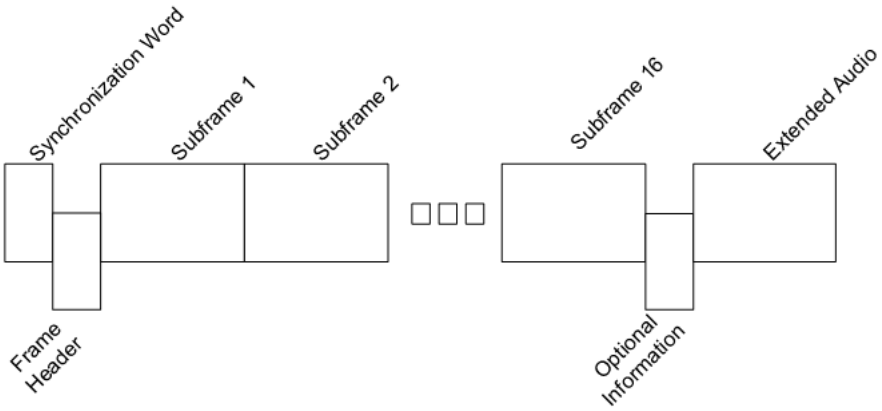


Figure 5-2: DTS frame structure

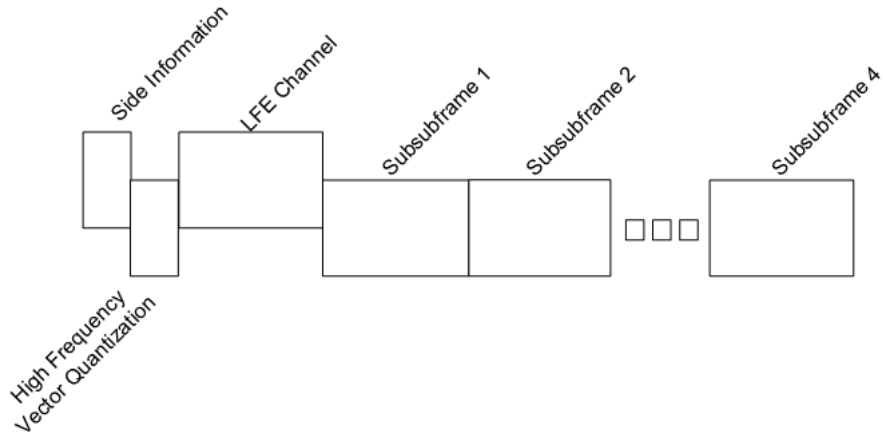


Figure 5-3: Subframe structure

5.3 Synchronization

DTS bit streams consist of a sequence of audio frames, usually of equal size and each frame begins with a 32-bit synchronization word.

The first decoding step is to search the input bit stream for SYNC. In order to reduce the probability of false synchronization, 6 bits after SYNC in the bit stream may be further checked, (FTYPE and SHORT), since they usually do not change for normal frames (they do carry useful information about frame structure). These 6 bits should be 0x3f (the binary 111111) for normal frames and are called synchronization word extension. Concatenating them with SYNC gives an extended synchronization word (32 + 6 = 38 bits).

5.4 Frame header

5.4.1 General Information about the Frame Header

The frame header consists of a bit stream header and a primary audio coding header. The bit stream header provides information about the construction of the frame, the encoder configuration such as core source sampling frequency and various optional operational features such as embedded dynamic range control. The primary audio coding header specifies the packing arrangement and coding formats used at the encoder to assemble the audio coding side information. Many elements in the headers are repeated for each separate audio channel.

5.4.2 Bit stream header

Table 5-1: Bit-stream header

Core Frame Header	Size (bits)
SYNC = ExtractBits(32); //0x7FFE8001	32
FTYPE = ExtractBits(1);	1
SHORT = ExtractBits(5);	5
CPF = ExtractBits(1);	1
NBLKS = ExtractBits(7);	7
FSIZE = ExtractBits(14);	14
AMODE = ExtractBits(6);	6
SFREQ = ExtractBits(4);	4
RATE = ExtractBits(5);	5
FixedBit = ExtractBit(1);	1
DYNF = ExtractBits(1);	1
TIMEF = ExtractBits(1);	1
AUXF = ExtractBits(1);	1
HDCD = ExtractBits(1);	1
EXT_AUDIO_ID = ExtractBits(3);	3
EXT_AUDIO = ExtractBits(1);	1
ASPF = ExtractBits(1);	1
LFF = ExtractBits(2);	2
HFLAG = ExtractBits(1);	1

if (CPF == 1) // Present only if CPF=1.
  HCRC = ExtractBits(16);
	16

FILTS = ExtractBits(1);	1
VERNUM = ExtractBits(4);	4
CHIST = ExtractBits(2);	2
PCMR = ExtractBits(3);	3
SUMF = ExtractBits(1);	1
SUMS = ExtractBits(1);	1

switch (VERNUM){
case 6:
  DIALNORM = ExtractBits(4);
  DNG = -(16+DIALNORM);
  break;
case7:
  DIALNORM = ExtractBits(4);
  DNG = -DIALNORM;
  break;
default:
  UNSPEC = ExtractBits(4);
  DNG = DIALNORM = 0;
  break;
}
	4

SYNC (Sync word)

The sync word denotes the start of an audio frame, or a component thereof. For a core substream, the sync word is 0x7ffe8001. A core frame can also exist in an extension substream, where the sync word is 0x02b9261.

FTYPE (Frame type)

This field indicates the type of current frame:

Table 5-2: Frame Type

FTYPE	Frame Type
1	Normal frame
0	Termination frame

Termination frames are used when it is necessary to accurately align the end of an audio sequence with a video frame end point. A termination block carries n×32 core audio samples where block length n is adjusted to just fall short of the video end point. Two termination frames may be transmitted sequentially to avoid transmitting one excessively small frame.

SHORT (Deficit Sample Count)

This field defines the number of core samples by which a termination frame falls SHORT of the normal length of a block. A block = 32 PCM core samples per channel, corresponding to the number of PCM core samples that are fed to the core filter bank to generate one subband sample for each subband. A normal frame consists of blocks of 32 PCM core samples, while a termination frame provides the flexibility of having a frame size precision finer than the 32 PCM core sample block. On completion of a termination frame, (SHORT+1) PCM core samples shall be padded to the output buffers of each channel. The padded samples may be zeros or they may be copies of adjacent samples.

Table 5-3: Deficit Sample Count

SHORT	Valid Value or Range of SHORT
1	[0,30]
0	31 (indicating a normal frame).

CPF (CRC Present Flag)

This field should always be set to 0. In the event that a bitstream was created prior to the occlusion of this bit from the bitstream definition, all decoders are required to test this flag.

NBLKS (Number of PCM Sample Blocks)

This field indicates that there are (NBLKS+1) blocks (a block = 32 PCM core samples per channel, corresponding to the number of PCM samples that are fed to the core filter bank to generate one subband sample for each subband) in the current frame (see note). The actual core encoding window size is 32 × (NBLKS+1) PCM samples per channel. Valid range for NBLKS: 5 to 127. Invalid range for NBLKS: 0 to 4. For normal frames, this indicates a window size of either 4 096, 2 048, 1 024, 512, or 256 samples per channel. For termination frames, NBLKS can take any value in its valid range.

NOTE: When frequency extension stream (X96) is present, the PCM core samples represent the samples at the output of the decimator that precedes the core encoder. This k-times decimator translates the original PCM source samples with the sampling frequency of Fs_src= k×SFREQ to the core PCM samples (Fs_core=SFREQ) suitable for the encoding by the core encoder. The core encoder can handle sampling frequencies SFREQ ≤ 48 kHz and consequently:

- k=2 for 48 kHz < Fsrc ≤ 96 kHz; and
- k=4 for 96 kHz < Fsrc ≤ 192 kHz

FSIZE (Primary Frame Byte Size)

(FSIZE+1) is the total byte size of the current frame including primary audio data as well as any extension audio data. Valid range for FSIZE: 95 to 16 383. Invalid range for FSIZE: 0 to 94.

AMODE (Audio Channel Arrangement)

Audio channel arrangement that describes the number of audio channels (CHS) and the audio playback arrangement (see Table 5-4). Unspecified modes may be defined at a later date (user defined code) and the control data required to implement them, i.e. channel assignments, down mixing etc., can be uploaded from the player platform.

Table 5-4: Audio channel arrangement (AMODE)

AMODE	CHS	Arrangement
0b000000	1	A
0b000001	2	A + B (dual mono)
0b000010	2	L + R (stereo)
0b000011	2	(L+R) + (L-R) (sum-difference)
0b000100	2	LT +RT (left and right total)
0b000101	3	C + L + R

AMODE	CHS	Arrangement
0b000110	3	L + R+ S
0b000111	4	C + L + R+ S
0b001000	4	L + R+ SL+SR
0b001001	5	C + L + R+ SL+SR
0b001010	6	CL + CR + L + R + SL + SR
0b001011	6	C + L + R+ LR + RR + OV
0b001100	6	CF+ CR + LF+ RF+LR + RR
0b001101	7	CL + C + CR + L + R + SL + SR
0b001110	8	CL + CR + L + R + SL1 + SL2+ SR1 + SR2
0b001111	8	CL + C+ CR + L + R + SL + S+ SR
0b010000 - 0b111111		User defined
NOTE: L = left, R = right, C = centre, S = surround, F = front, R = rear, T = total, OV = overhead, A = first mono channel, B = second mono channel.		

SFREQ (Core Audio Sampling Frequency)

This field specifies the sampling frequency of audio samples in the core encoder, based on Table 5-5. When the source sampling frequency is beyond 48 kHz the audio is encoded in up to 3 separate frequency bands. The base-band audio, for example, 0 kHz to 16 kHz, 0 kHz to 22,05 kHz or 0 kHz to 24 kHz, is encoded and packed into the core audio data arrays. The SFREQ corresponds to the sampling frequency of the base-band audio. The audio above the base-band (the extended bands), for example, 16 kHz to 32 kHz, 22,05 kHz to 44,1 kHz, 24 kHz to 48 kHz, is encoded and packed into the extended coding arrays which reside at the end of the core audio data arrays. If the decoder is unable to make use of the high sample rate data this information may be ignored and the base-band audio converted normally using a standard sampling rates (32 kHz, 44,1 kHz or 48 kHz). If the decoder is receiving data coded at sampling rates lower than that available from the system then interpolation (2× or 4×) will be required (see Table 5-6).

Table 5-5: Core audio sampling frequencies

SFREQ	Source Sampling Frequency	TimeStamp SampleRate
0b0000	Invalid	
0b0001	8 kHz	8 000
0b0010	16 kHz	16 000
0b0011	32 kHz	32 000
0b0100	Invalid	
0b0101	Invalid	
0b0110	11,025 kHz	11 025
0b0111	22,05 kHz	22 050
0b1000	44,1 kHz	44 100
0b1001	Invalid	
0b1010	Invalid	
0b1011	12 kHz	12 000
0b1100	24 kHz	24 000
0b1101	48 kHz	48 000
0b1110	Invalid	
0b1111	Invalid	

Table 5-6: Sub-sampled audio decoding for standard sampling rates

Core Audio Sampling Frequency	Hardware Sampling Frequency	Required Filtering
8 kHz	32 kHz	4 × Interpolation
16 kHz	32 kHz	2 × Interpolation
32 kHz	32 kHz	none
11 kHz	44,1 kHz	4 × Interpolation
22,05 kHz	44,1 kHz	2 × Interpolation
44,1 kHz	44,1 kHz	none
12 kHz	48 kHz	4 × Interpolation
24 kHz	48 kHz	2 × Interpolation
48 kHz	48 kHz	none

RATE (Transmission Bit Rate)

RATE specifies the targeted transmission data rate for the current frame of audio (see Table 5-7). The open mode allows for bit rates not defined by the table.

Table 5-7: RATE parameter versus targeted bit-rate

RATE	Targeted Bit Rate [Kbits/s]
0b00000	32
0b00001	56
0b00010	64
0b00011	96
0b00100	112
0b00101	128
0b00110	192
0b00111	224
0b01000	256
0b01001	320
0b01010	384
0b01011	448
0b01100	512
0b01101	576
0b01110	640
0b01111	768
0b10000	960
0b10001	1 024
0b10010	1 152
0b10011	1 280
0b10100	1 344
0b10101	1 408
0b10110	1 411,2
0b10111	1 472
0b11000	1 536
0b11101	open
other codes	invalid

The targeted transmission rate may be greater than or equal to the actual bit rate.

FixedBit (reserved field)

This field is always set to 0.

DYNF (Embedded Dynamic Range Flag)

DYNF indicates if embedded dynamic range coefficients are included at the start of each subframe. Dynamic range correction may be implemented on all channels using these coefficients for the duration of the subframe.

Table 5-8: Embedded Dynamic Range Flag

DYNF	Dynamic Range Coefficients
0	not present
1	present

TIMEF (Embedded Time Stamp Flag)

This field indicates if embedded time stamps are included at the end of the core audio data.

Table 5-9: Embedded Time Stamp Flag

TIMEF	Time Stamps
0	not present
1	present

AUXF (Auxiliary Data Flag)

This field indicates if auxiliary data bytes are appended at the end of the core audio data.

Table 5-10: Auxiliary Data Flag

AUXF	Auxiliary Data Bytes
0	not present
1	present

HDCD

The source material is mastered in HDCD format if HDCD = 1 and otherwise HDCD = 0.

EXT_AUDIO_ID (Extension Audio Descriptor Flag)

This flag has meaning only if the EXT_AUDIO = 1 (see Table 5-11) and then it indicates the type of data that has been placed in the extended audio.

Table 5-11: Extension Audio Descriptor Flag

EXT_AUDIO_ID	Type of Extension Data
0	Channel Extension (XCH)
1	Reserved
2	Frequency Extension (X96)
3	Reserved
4	Reserved
5	Reserved
6	Channel Extension (XXCH)
7	Reserved

EXT_AUDIO (Extended Coding Flag)

Indicates if extended audio coding data are present after the core audio data. Extended audio data will include the data for the extended bands of the 5 normal primary channels as well as all bands of additional audio channels. To simplify the process of implementing a 5.1 channel 48 kHz decoder, the extended coding data arrays are placed at the end of the core audio array.

Table 5-12: Extended Coding Flag

EXT_AUDIO	Extended Audio Data
0	not present
1	present

ASPF (Audio Sync Word Insertion Flag)

Indicates how often the audio data check word DSYNC (0xFFFF Extension Audio Descriptor Flag) occurs in the data stream. DSYNC is used as a simple means of detecting the presence of bit errors in the bit stream and is used as the final data verification stage prior to transmitting the reconstructed PCM words to the DACs.

Table 5-13: Audio Sync Word Insertion Flag

ASPF	DSYNC Placed at End of Each
0	Subframe
1	Subsubframe

LFF (Low Frequency Effects Flag)

Indicates if the LFE channel is present and the choice of the interpolation factor to reconstruct the LFE channel (see Table 5-14).

Table 5-14: Flag for LFE channel

LFF	LFE Channel	Interpolation Factor
0	not present	
1	Present	128
2	Present	64
3	Invalid	

HFLAG (Predictor History Flag Switch)

If frames are to be used as possible entry points into the data stream or as audio sequence\start frames the ADPCM predictor history may not be contiguous. Hence these frames can be coded without the previous frame predictor history, making audio ramp-up faster on entry. When generating ADPCM predictions for current frame, the decoder will use reconstruction history of the previous frame if HFLAG = 1. Otherwise, the history will be ignored.

HCRC (Header CRC Check Bytes)

If CPF = 1 then HCRC shall be extracted from the bitstream. The CRC value test shall not be applied.

FILTS (Multirate Interpolator Switch)

Indicates which set of 32-band interpolation FIR coefficients is to be used to reconstruct the subband audio (see Table 5-15).

Table 5-15: Multirate interpolation filter bank switch

FILTS	32-band Interpolation Filter
0	Non-perfect Reconstruction
1	Perfect Reconstruction

VERNUM (Encoder Software Revision)

VERNUM indicates of the revision status of the encoder software (see Table 5-16). In addition, it is used to indicate the presence of the dialog normalization parameters (see Table 5-20).

Table 5-16: Encoder software revision

VERNUM	Encoder Software Revision
0 to 6	Future revision (compatible with the present document)
7	Current
8 to 15	Future revision (incompatible with the present document)
NOTE: If the decoder encounters the DTS stream with the VERNUM>7 and the decoder is not designed for that specific encoder software revision then it shall mute its outputs.	

CHIST (Copy History)

This field indicates the copy history of the audio. Because of the copyright regulations, the exact definition of this field is deliberately omitted.

PCMR (Source PCM Resolution)

This field indicates the quantization resolution of source PCM samples (see Table 5-17). The left and right surrounding channels of the source material are mastered in DTS ES format if ES = 1 and otherwise if ES = 0.

Table 5-17: Quantization resolution of source PCM samples

PCMR	Source PCM Resolution	ES
0b000	16 bits	0
0b001	16 bits	1
0b010	20 bits	0
0b011	20 bits	1
0b110	24 bits	0
0b101	24 bits	1
Others	Invalid	invalid

SUMF (Front Sum/Difference Flag)

This parameter indicates if front left and right channels are sum-difference encoded prior to encoding (see Table 5-18). If set to zero no decoding post processing is required at the decoder.

Table 5-18: Sum/difference decoding status of front left and right channels

SUMF	Front Sum/Difference Encoding
0	L = L, R = R
1	L = L + R, R = L - R

SUMS (Surrounds Sum/Difference Flag)

This parameter indicates if left and right surround channels are sum-difference encoded prior to encoding (see Table 5-19). If set to zero no decoding post processing is required at the decoder.

Table 5-19: Sum/difference decoding status of left and right surround channels

SUMS	Surround Sum/Difference Encoding
0	Ls = Ls, Rs = Rs
1	Ls = Ls + Rs, Rs = Ls - Rs

DIALNORM/UNSPEC (Dialog Normalization /Unspecified)

For the values of VERNUM = 6 or 7 this 4-bit field is used to determine the dialog normalization parameter. For all other values of the VERNUM this field is a place holder that is not specified at this time.

The Dialog Normalization Gain (DNG), in dB, is specified by the encoder operator and is used to directly scale the decoder outputs samples. In the DTS stream the information about the DNG value is transmitted by means of combined data in the VERNUM and DIALNORM fields (see Table 5-20).

For all other values of the VERNUM (i.e. 0, 1, 2, 3, 4, 5, 8, 9, …15) the UNSPEC 4-bit field should be extracted but ignored by the decoder. In addition, for these VERNUM values, the Dialog Normalization Gain should be set to 0 i.e. DNG=0 indicates No Dialog Normalization.

Table 5-20: Dialog Normalization Parameter

DIALNORM	VERNUM	DNG (dB)
0	7	0
1	7	-1
2	7	-2
3	7	-3
4	7	-4
5	7	-5
6	7	-6
7	7	-7
8	7	-8
9	7	-9
10	7	-10
11	7	-11
12	7	-12
13	7	-13

DIALNORM	VERNUM	DNG (dB)
14	7	-14
15	7	-15
0	6	-16
1	6	-17
2	6	-18
3	6	-19
4	6	-20
5	6	-21
6	6	-22
7	6	-23
8	6	-24
9	6	-25
10	6	-26
11	6	-27
12	6	-28
13	6	-29
14	6	-30
15	6	-31

5.4.3 Primary Audio Coding Header

Table 5-21: Primary audio coding header

Core audio coding header	Size (bits)
`SUBFS = ExtractBits(4);` `nSUBFS = SUBFS + 1;`	4
`PCHS = ExtractBits(3);` `nPCHS = PCHS + 1;`	3

`for (ch=0; ch<nPCHS; ch++) {`
`  SUBS[ch] = ExtractBits(5);`
`  nSUBS[ch] = SUBS[ch] + 2;`
`}`
	5 bits per channel

`for (ch=0; ch<nPCHS; ch++) {`
`  VQSUB[ch] = ExtractBits(5);`
`  nVQSUB[ch] = VQSUB[ch] + 1;`
`}`
	5 bits per channel

`for (ch=0; ch<nPCHS; ch++) {`
`  JOINX[ch] = ExtractBits(3);`
`}`
	3 bits per channel

`for (ch=0; ch<nPCHS; ch++) {`
`  THUFF[ch] = ExtractBits(2);`
`}`
	2 bits per channel

`for (ch=0; ch<nPCHS; ch++) {`
`  SHUFF[ch] = ExtractBits(3);`
`}`
	3 bits per channel

`for (ch=0; ch<nPCHS; ch++) {`
`  BHUFF[ch] = ExtractBits(3);`
`}`
	3 bits per channel

`// ABITS=1:`
`n=0;`
`for (ch=0; ch<nPCHS; ch++)`
`  SEL[ch][n] = ExtractBits(1);`
`// ABITS = 2 to 5:`
`for (n=1; n<5; n++)`
`  for (ch=0; ch<nPCHS; ch++)`
`  SEL[ch][n] = ExtractBits(2);`
`// ABITS = 6 to 10:`
`for (n=5; n<10; n++)`
`  for (ch=0; ch<nPCHS; ch++)`
`  SEL[ch][n] = ExtractBits(3);`
`// ABITS = 11 to 26:`
`for (n=10; n<26; n++)`
`  for (ch=0; ch<nPCHS; ch++)`
`  SEL[ch][n] = 0; // Not transmitted, set to zero.`
	variable bits

`n = 0;  // ABITS = 1 :`
`for (ch=0; ch<nPCHS; ch++)`
`  if ( SEL[ch][n] == 0 ) { // Transmitted only if SEL=0 (Huffman code used)`
`  // Extract ADJ index`
	2 bits per occasion

Core audio coding header	Size (bits)

ADJ = ExtractBits(2);
// Look up ADJ table
arADJ[ch][n] = AdjTable[ADJ];
}
for (n=1; n<5; n++){ // ABITS = 2 to 5:
for (ch=0; ch<nPCHS; ch++){
if (SEL[ch][n] < 3) { // Transmitted only when SEL<3
// Extract ADJ index
ADJ = ExtractBits(2);
// Look up ADJ table
arADJ[ch][n] = AdjTable[ADJ];
}
}
}
for (n=5; n<10; n++){ // ABITS = 6 to 10:
for (ch=0; ch<nPCHS; ch++){
if (SEL[ch][n] < 7) { // Transmitted only when SEL<7
// Extract ADJ index
ADJ = ExtractBits(2); Look up ADJ table
arADJ[ch][n] = AdjTable[ADJ]; //
}
}
}
if (CPF==1) // Present only if CPF=1
AHCRC = ExtractBits(16);
	16

SUBFS (Number of Subframes)

This field indicates that there are nSUBFS = SUBFS+1 audio subframes in the core audio frame. SUBFS is valid for all audio channels.

PCHS (Number of Primary Audio Channels)

This field indicates that there are nPCHS = PCHS+1 $\leq$ 5 primary audio channels in the current frame. If AMODE flag indicates more than five channels apart from LFE, the additional channels are the extended channels and are packed separately in the extended data arrays.

SUBS (Subband Activity Count)

This field indicates that there are nSUBS[ch] = SUBS[ch]+2 active subbands in the audio channel ch. Samples in subbands above nSUBS[ch] are zero, provided that intensity coding in that subband is disabled.

VQSUB (High Frequency VQ Start Subband)

This field indicates that high frequency samples starting from subband nVQSUB[ch]=VQSUB[ch]+1 are VQ encoded. High frequency VQ is used only for high frequency subbands, but it may go down to low frequency subbands for such audio episodes as silence. In case of insufficient MIPS, the VQs for the highest frequency subbands may be ignored without causing audible distortion.

JOINX (Joint Intensity Coding Index)

JOINX[ch] indicates if joint intensity coding is enabled for channel ch and which audio channel is the source channel from which channel ch will copy subband samples (Table 5-22). It is assumed that the source channel index is smaller than that of the intensity channel.

Table 5-22: Joint Subband Coding Status and Source Channels

JOINX[ch]	Joint Intensity	Source Channel
0	Disabled	
> 0	Enabled	JOINX[ch]

THUFF (Transient Mode Code Book)

This field indicates which Huffman codebook was used to encode the transient mode data (see Table 5-23).

Table 5-23: Selection of Huffman Codebook for Encoding TMODE

THUFF[ch]	Huffman Codebook
0	A4
1	B4
2	C4
3	D4

SHUFF (Scale Factor Code Book)

The scale factors of a channel are quantized nonlinearly using either a 6-bit (64-level, 2,2 dB per step) or a 7-bit (128-level, 1,1 dB per step) square root table, depending on the application. The quantization indexes may be further compressed by one of the five Huffman codes and this information is transmitted to the decoder by SHUFF[ch] (see Table 5-24).

Table 5-24: Code Books and Square Root Tables for Scale Factors

SHUFF[ch]	Code Book	Square Root Table
0	SA129	6 bit (clause D.1.1)
1	SB129	6 bit (clause D.1.1)
2	SC129	6 bit (clause D.1.1)
3	SD129	6 bit (clause D.1.1)
4	SE129	6 bit (clause D.1.1)
5	6-bit linear	6 bit (clause D.1.1)
6	7-bit linear	7 bit (clause D.1.2)
7	Invalid	Invalid

BHUFF (Bit Allocation Quantizer Select)

Indicates the codebook that was used to encode the bit allocation index ABITS (to be transmitted later) (see Table 5-25).

Table 5-25: Codebooks for Encoding Bit Allocation Index ABITS

BHUFF[ch]	Codebook (clause D.5.6)
0	A12
1	B12
2	C12
3	D12
4	E12
5	Linear 4-bit
6	Linear 5-bit
7	Invalid

SEL (Quantization Index Codebook Select)

After subband samples are quantized using a mid-tread linear quantizer, the quantization indexes may be further encoded using either entropy (Huffman) or block coding in order to reduce bit rate. Therefore, the subband samples may appear in the bitstream as plain quantization indexes (no further encoding), entropy (Huffman) codes, or block codes. For channel ch, the selection of a particular codebook for a mid-tread linear quantizer indexed by ABITS[ch] is transmitted to the decoder as SEL[ch][ABITS[ch]]. No SEL is transmitted for ABITS[ch]≥11, because no further encoding is used for those quantizers. The decoder can find out the particular codebook that was used using ABITS[ch] and SEL[ch][ABITS[ch]] to look up Table 5-26.

Table 5-26: Selection of Quantization Levels and Codebooks

Quantizer Index (ABITS)	Number of Index Quantization Levels	Codebook Select (SEL)							
		0	1	2	3	4	5	6	7
0	0	Not transmitted							
1	3	A3	V3						
2	5	A5	B5	C5	V5				
3	7	A7	B7	C7	V7				
4	9	A9	B9	C9	V9				
5	13	A13	B13	C13	V13				
6	17	A17	B17	C17	D17	E17	F17	G17	V17
7	25	A25	B25	C25	D25	E25	F25	G25	V25
8	33 or 32	A33	B33	C33	D33	E33	F33	G33	NFE
9	65 or 64	A65	B65	C65	D65	E65	F65	G65	NFE
10	129 or 128	A129	B129	C129	D129	E129	F129	G129	NFE
11	256	NFE							
12	512	NFE							
13	1 024	NFE							
14	2 048	NFE							
15	4 096	NFE							
16	8 192	NFE							
17	16 384	NFE							
18	32 768	NFE							
19	65 536	NFE							
20	131 072	NFE							
21	262 144	NFE							
22	524 288	NFE							
23	1 048 576	NFE							
24	2 097 152	NFE							
25	4 194 304	NFE							
26	8 388 608	NFE							
27-32	Invalid	Invalid							

NOTE: NFE = No further encoding is used to encode the linearly quantized subband samples. A,B,C,D,E,F,G = Subband samples are encoded by Huffman code. V = 4 subband samples are grouped and encoded using 4-element block code.

ADJ (Scale Factor Adjustment Index)

A scale factor adjustment index is transmitted whenever a SEL value indicates a Huffman codebook. This index points to the adjustment values shown in Table 5-27. This adjustment value should be multiplied by the scale factor (SCALE).

Table 5-27: Scale Factor Adjustment Index

Scale Factor Adjustment Index (ADJ)	Adjustment Value
0	1,0000
1	1,1250
2	1,2500
3	1,4375

NOTE: This table shows the scale factor adjustment index values if Huffman coding is used to encode the subband quantization indexes.

AHCRC (Audio Header CRC Check Word)

If CPF = 1 then AHCRC shall be extracted from the bitstream. The CRC value test shall not be applied.

5.5 Unpack Subframes

5.5.1 Primary Audio Coding Side Information

Table 5-28: Core side information

Primary audio side information	Size (bits)
SSC = ExtractBits(2); nSSC = SSC+1;	2
PSC = ExtractBits(3);	3

for (ch=0; ch<nPCHS; ch++)
  for (n=0; n<nSUBS[ch]; n++)
  PMODE[ch][n] = ExtractBits(1);
	1 bit per subband

for (ch=0; ch<nPCHS; ch++)
for (n=0; n<nSUBS[ch]; n++)
  if (PMODE[ch][n]>0) { // Transmitted only when ADPCM active
  // Extract the VQindex
  nVQIndex = ExtractBits(12);
  // Look up the VQ table for prediction coefficients.
  ADPCMCoeffVQ.LookUp(nVQIndex, PVQ[ch][n]) // 4 coefficients
  }
	12 bits per occurrence

for (ch=0; ch<nPCHS; ch++) {
  // BHUFF tells which codebook was used
  nQSelect = BHUFF[ch];
  // Use this codebook to decode the bit stream for ABITS[ch][n]
  for (n=0; n<nVQSUB[ch]; n++) // Not for VQ encoded subbands.
  QABITS.ppQ[nQSelect]->InverseQ(InputFrame, ABITS[ch][n])
  }
	variable bits

for (ch=0; ch<nPCHS; ch++){
  for (n=0; n<NumSubband; n++)
  TMODE[ch][n] = 0;
  // Decode TMODE[ch][n]
  if (nSSC>1) {// Transient possible only if more than one subsubframe.
  for (ch=0; ch<nPCHS; ch++) {
  // TMODE[ch][n] is encoded by a codebook indexed by THUFF[ch]
  nQSelect = THUFF[ch];
  for (n=0; n<nVQSUB[ch]; n++) // No VQ encoded subbands
  if (ABITS[ch][n] >0) // Present only if bits allocated
  // Use codebook nQSelect to decode TMODE from the bit stream
  QTMODE.ppQ[nQSelect]->InverseQ(InputFrame,TMODE[ch][n])
  }
  }
}
	variable bits

for (ch=0; ch<nPCHS; ch++){
 	
  // Clear SCALES
 	
  for (n=0; n<NumSubband; n++){
  SCALES[ch][n][0] = 0;
  SCALES[ch][n][1] = 0;
  }
  nQSelect = SHUFF[ch]; // SHUFF indicates which codebook was used to encode SCALES
  // Select the root square table (SCALES were nonlinearly quantized).
  if (nQSelect == 6)
  pScaleTable = &RMS7Bit; // 7-bit root square table
  else
  pScaleTable = &RMS6Bit; // 6-bit root square table
  // Clear accumulation (if Huffman code was used, the difference of SCALES was encoded).
  nScaleSum = 0;
  // Extract SCALES for Subbands up to VQSUB[ch]
  for (n=0; n<nVQSUB[ch]; n++)
  if (ABITS[ch][n] >0) { // Not present if no bit allocated
  // First scale factor
  // Use the (Huffman) code indicated by nQSelect to decode the quantization
  // index of SCALES from the bit stream
  QSCALES.ppQ[nQSelect]->InverseQ(InputFrame, nScale);
  // Take care of difference encoding
  if (nQSelect < 5) // Huffman encoded, nScale is the difference
  nScaleSum += nScale; // of the quantization indexes of SCALES.
  else // Otherwise, nScale is the quantization
  nScaleSum = nScale; // level of SCALES.
	variable bits

Primary audio side information	Size (bits)

`// Look up SCALES from the root square table`
`pScaleTable->LookUp(nScaleSum, SCALES[ch][n][0])`
`// Two scale factors transmitted if there is a transient`
`if (TMODE[ch][n]>0) {`
`// Use the (Huffman) code indicated by nQSelect to decode the`
`// quantization index of SCALES from the bit stream`
`QSCALES.ppQ[nQSelect]->InverseQ(InputFrame, nScale);`
`// Take care of difference encoding`
`if ( nQSelect < 5 ) // Huffman encoded, nScale is the`
`nScaleSum += nScale; // of SCALES.`
`else // Otherwise, nScale is SCALES`
`nScaleSum = nScale; // itself.`
`// Look up SCALES from the root square table`
`pScaleTable->LookUp(nScaleSum, SCALES[ch][n][1])`
`}`
`}`
`//`
`// High frequency VQ subbands`
`//`
`for (n=nVQSUB[ch]; n<nSUBS[ch]; n++) {`
`// Use the code book indicated by nQSelect to decode`
`// the quantization index of SCALES from the bit stream`
`QSCALES.ppQ[nQSelect]->InverseQ(InputFrame, nScale);`
`// Take care of difference encoding`
`if ( nQSelect < 5 ) // Huffman encoded, nScale is the`
`nScaleSum += nScale; // of SCALES.`
`else // Otherwise, nScale is SCALES`
`nScaleSum = nScale; // itself.`
`// Look up SCALES from the root square table`
`pScaleTable->LookUp(nScaleSum, SCALES[ch][n][0])`
`}`
`}`	

`for (ch=0; ch<nPCHS; ch++)`
`if (JOINX[ch]>0 ) // Transmitted only if joint subband coding enabled.`
`JOIN_SHUFF[ch] = ExtractBits(3);`
	3 bits per channel

`for (ch=0; ch<nPCHS; ch++)`
`if (JOINX[ch]>0 ) { // Only if joint subband coding enabled.`
`nSourceCh = JOINX[ch]-1; // Get source channel. JOINX counts`
`// channels as 1,2,3,4,5, so minus 1.`
`nQSelect = JOIN_SHUFF[ch]; // Select code book.`
`for (n=nSUBS[ch]; n<nSUBS[nSourceCh]; n++){`
`// Use the code book indicated by nQSelect to decode`
`// the quantization index of JOIN_SCALES`
`QSCALES.ppQ[nQSelect]->InverseQ(InputFrame, nJScale);`
`// Bias by 64`
`nJScale = nJScale + 64;`
`// Look up JOIN_SCALES from the joint scale table;`
`JScaleTbl.LookUp(nJScale, JOIN_SCALES[ch][n]);`
`}`
`}`
	variable bits

`if ( DYNF != 0 ) {`
 	
`nIndex = ExtractBits(8);`
 	
`RANGEtbl.LookUp(nIndex,RANGE);`
 	
`// The following range adjustment is to be performed`
 	
`// after QMF reconstruction`
 	
`for (ch=0; ch<nPCHS; ch++)`
`for (n=0; n<nNumSamples; n++)`
`AudioCh[ch].ReconstructedSamples[n] *= RANGE;`
`}`
	8

`if ( CPF==1 ) // Present only if CPF=1.` `SICRC = ExtractBits(16);`	16

SSC (Subsubframe Count)

Indicates that there are nSSC = SSC+1 subsubframes in the current audio subframe.

PSC (Partial Subsubframe Sample Count)

PSC indicates the number of subband samples held in a partial subsubframe for each of the active subbands. A partial subsubframe is one which has less than 8 subband samples. It exists only in a termination frame and is always at the end of last normal subsubframe. A DSYNC word will always occur after a partial subsubframe.

PMODE (Prediction Mode)

PMODE[ch][n]=1 indicates that ADPCM prediction is used (active) for subband n of primary audio channel [ch] and PMODE[ch][n]=0 otherwise. ADPCM shall be extracted from the bit stream for all subbands, but ADPCM reconstruction can be limited to the lowest 20 subbands if DSP does not have enough MIPS.

PVQ (Prediction Coefficients VQ Address)

Indexes to the vector code book (clause D.10.1) to obtain ADPCM prediction coefficients. PVQ is transmitted only for subbands where ADPCM is active.

ABITS (Bit Allocation Index)

ABITS[ch][n] is the index to the mid-tread linear quantizer that was used to quantize the subband samples for the n^{th} subband of channel ch. ABITS[ch][n] may be transmitted as either a 4-bit or 5-bit word. In the case of a 4-bit word, it may be further encoded using one of the 5 Huffman codes. This encoding is the same for all subbands of each channel and is conveyed by BHUFF as shown in Table 5-25. There is obviously no need to allocate bits for the high frequency subbands because they are encoded using VQ.

TMODE (Transition Mode)

TMODE[ch][n] indicates if there is a transient inside a subframe (subband analysis window) for subband n of channel ch. If there is a transient (TMODE[ch][n]>0), it further indicates that the transition occurred in subsubframe (subband analysis subwindow) TMODE[ch][n] + 1. TMODE[ch][n] is encoded by one of the 4 Huffman codes and the selection of which is conveyed by THUFF (see Table 5-23). The decoder assumes that there is no transition (TMODE[ch][n]=0) for all subbands of all channels unless it is told otherwise by the bit stream. Transient does not occur in the following situations, so TMODE is not transmitted when:

- Only one subsubframe within the current subframe. This is because the time resolution of transient analysis is a subsubframe (subband analysis subwindow).
- VQ encoded high frequency subbands. If there is a transient for a subband, it would not have been VQ encoded.
- Subbands without bit allocation. If no bits are allocated for a subband, there is no need for transient.

SCALES (Scale Factors)

One scale factor is transmitted for subbands without transient. Otherwise two are transmitted, one for the episode before the transient and the other for after the transient. The quantization indexes of the scale factors may be encoded by Huffman code as shown in Table 5-24. If this is the case, they are difference-encoded before Huffman coding. The scale factors are finally obtained by using the quantization indexes to look up either the 6-bit or 7-bit square root quantization table according to Table 5-24.

JOIN SHUFF (Joint Subband Codebook Select)

If joint subband coding is enabled (JOINX[ch]>0), JOIN SHUFF[ch] selects which code book was used to encode the scale factors (JOIN SCALES) which will be used when copying subband samples from the source channel to the current channel ch. These scale factors are encoded in exactly the same way as that for SCALES, using Table 5-24 to look up the codebook.

JOIN SCALES (Scale Factors for Joint Subband Coding)

The scale factors are used to scale the subband samples copied from the source channel (JOINX[ch]-1) to the current channel. The index of the scale factor is encoded using the code book indexed by JOIN SHUFF[ch]. After this index is decoded, it is used to look up the table in clause D.4, to get the scale factor. No transient is permitted for jointly encoded subbands, so a single scale factor is included. The joint subbands start from the nSUBS of the current channel until the nSUBS of the source channel.

RANGE (Dynamic Range Coefficient)

Dynamic range coefficient is to allow for the convenient compression of the audio dynamic range at the output of the decoder. Dynamic range compression is particularly important in listening environments where high ambient noise levels make it impossible to discriminate low level signals without risking damaging the loudspeakers during loud passages. This problem is further compounded by the growing use of 20-bit PCM audio recordings which exhibit dynamic ranges as high as 110 dB.

Each coefficient is 8-bit signed fractional Q2 binary and represents a logarithmic gain value as shown in clause D.4 giving a range of ±31,75 dB in steps of 0,25 dB. Dynamic range compression is affected by multiplying the decoded audio samples by the linear coefficient.

The degree of compression can be altered with the appropriate adjustment to the coefficient values at the decoder and can be switched off completely by ignoring the coefficients.

SICRC (Side Information CRC Check Word)

If CPF = 1 then SICRC shall be extracted from the bitstream. The CRC value test shall not be applied.

5.6 Primary Audio Data Arrays

Table 5-29: Core audio data arrays

Primary audio data	Size (bits)

`for (ch=0; ch<nPCHS; ch++)`
`  for (n=nVQSUB[ch]; n<nSUBS[ch]; n++) {`
`    // Extract the VQ address from the bit stream`
`    nVQIndex = ExtractBits(10);`
`    // Look up the VQ code book for 32 subband samples.`
`    HFreqVQ.LookUp(nVQIndex, HFREQ[ch][n])`
`    // Scale and take the samples`
`    Scale = (real)SCALES[ch][n][0]; // Get the scale factor`
`    for (m=0; m<nSSC*8; m++, nSample++)`
`      aPrmCh[ch].aSubband[n].raSample[m] = rScale*HFREQ[ch][n][m];`
`  }`
`if ( LFF>0 ) { // Present only if flagged by LFF`
`  // extract LFE samples from the bit stream`
`  for (n=0; n<2*LFF*nSSC; n++)`
`  LFE[n] = (signed int)(signed char)ExtractBits(8);`
`  // Use char to get sign extension because it is 8-bit 2's compliment.`
`  // Extract scale factor index from the bit stream`
`  LFEscaleIndex = ExtractBits(8);`
`  // Look up the 7-bit root square quantization table`
`  pLFE_RMS->LookUp(LFEscaleIndex,nScale);`
`  // Account for the quantizer step size which is 0.035`
`  rScale = nScale*0.035;`
`  // Get the actual LFE samples`
`  for (n=0; n<2*LFF*nSSC; n++)`
`    LFECh.rLFE[k] = LFE[n]*rScale;`
`  // Interpolation LFE samples`
`  LFECh.InterpolationFIR(LFF); // LFF indicates which interpolation filter to use`
`}`
	10 bits per subband
	
	8 bits per sample

Audio Data	

`// Select quantization step size table`
`if ( RATE == 0x1f )`
`  pStepSizeTable = &StepSizeLossLess;   // Lossless quantization`
`else`
`  pStepSizeTable = &StepSizeLossy;  // Lossy`
`// Unpack the subband samples`
`for (nSubSubFrame=0; nSubSubFrame<nSSC; nSubSubFrame++) {`
`  for (ch=0; ch<nPCHS; ch++)`
`    for (n=0; n<nVQSUB[ch]; n++) { // Not high frequency VQ subbands`
`      // Select the mid-tread linear quantizer`
`      nABITS = ABITS[ch][n];     // Select the mid-tread quantizer`
`      pCQGroup = &pCQGroupAUDIO[nABITS-1];// Select the group of`
`      // code books corresponding to the`
`      // the mid-tread linear quantizer.`
`      nNumQ = pCQGroupAUDIO[nABITS-1].nNumQ-1;// Number of code`
`      // books in this group`
`      // Determine quantization index code book and its type`
`      // Select quantization index code book`
`      nSEL = SEL[ch][nABITS-1];`
	variable bits

```
      // Determine its type
      nQType = 1;    // Assume Huffman type by default
      if ( nSEL==nNumQ ) {  // Not Huffman type
        if ( nABITS<=7 )
         nQType = 3;     // Block code
        else
         nQType = 2;     // No further encoding
      }
      if ( nABITS==0 )  // No bits allocated
        nQType = 0;
      // Extract bits from the bit stream
      switch ( nQType ) {
      case 0:  // No bits allocated
        for (m=0; m<8; m++)
        AUDIO[m] = 0;
        break;
      case 1:  // Huffman code
        for (m=0; m<8; m++)
        pCQGroup->ppQ[nSEL]->InverseQ(InputFrame, AUDIO[m]);
        break;
      case 2: // No further encoding
        for (m=0; m<8; m++) {
         // Extract quantization index from the bit stream
         pCQGroup->ppQ[nSEL]->InverseQ(InputFrame, nCode)
         // Take care of 2's compliment
          AUDIO[m] = pCQGroup->ppQ[nSEL]->SignExtension(nCode);
        }
        break;
      case 3: // Block code
        pCBQ = &pCBlockQ[nABITS-1]; // Select block code book
        m = 0;
        for (nBlock=0; nBlock<2; nBlock++) {
         // Extract the block code index from the bit stream
         pCQGroup->ppQ[nSEL]->InverseQ(InputFrame, nCode)
         // Look up 4 samples from the block code book
         pCBQ->LookUp(nCode,&AUDIO[m])
         m += 4;
        }
        break;
      default: // Undefined
        printf("ERROR: Unknown AUDIO quantization index code book.");
      }
    }
    // Account for quantization step size and scale factor
    // Look up quantization step size
    nABITS = ABITS[ch][n];
    pStepSizeTable->LookUp(nABITS, rStepSize);
    // Identify transient location
    nTmode = TMODE[ch][n];
    if ( nTmode == 0 ) // No transient
      nTmode = nSSC;
    // Determine proper scale factor
    if (nSubSubFrame<nTmode) // Pre-transient
      rScale = rStepSize * SCALES[ch][n][0]; // Use first scale factor
    else  // After-transient
      rScale = rStepSize * SCALES[ch][n][1]; // Use second scale factor
    // Adjustmemt of scale factor
    rScale *= arADJ[ch][SEL[ch][nABITS-1]];     // arADJ[][] are assumed 1
    // unless changed by bit
    // stream when SEL indicates
    // Huffman code.
    // Scale the samples
    nSample = 8*nSubSubFrame; // Set sample index
    for (m=0; m<8; m++, nSample++)
      aPrmCh[ch].aSubband[n].aSample[nSample] = rScale*AUDIO[m];
    // Inverse ADPCM
    if ( PMODE[ch][n] != 0 ) // Only when prediction mode is on.
      aPrmCh[ch].aSubband[n].InverseADPCM();
    // Check for DSYNC
    if ( (nSubSubFrame==(nSSC-1)) || (ASPF==1) ) {
      DSYNC = ExtractBits(16);
      if ( DSYNC != 0xffff )
        printf("DSYNC error at end of subsubframe #%d", nSubSubFrame);
    }
}
```

HFREQ (VQ Encoded Subbands)

At low bit rates, some high frequency subbands are encoded using Vector Quantization (VQ). Each vector from this code book consists of 32 subband samples, corresponding to the maximum possible subframe (4 normal subsubframes):

4 subsubframe × 8 samples/subsubframe = 32 samples:

If the current subframe is short of 32 samples, the remaining samples are padded with either zeros or "don't care" and then vector-quantized. The vector address is then included in the bit stream. After the decoder picks up the vector address, it looks up the vector code book to get the 32 samples. But the decoder will only pick nSSC x 8 out of the 32 samples and scale them with the scale factor SCALES.

LFE (Low Frequency Effect Data)

The presence of a LFE channel and its interpolation filter selection are flagged by LFF in the frame header (see Table 5-14). The number of decimated LFE samples in the current subframe is 2 × LFF × nSSC, corresponding to the decimation factor and the subframe size. The LFE samples are normalized with a scale factor and then quantized with a step size of 0,035, before being included in the bit stream as 8-bit 2's compliment. This scale factor is nonlinearly quantized using the 7-bit root square and then directly included in the bit stream right after the decimated LFE samples. Therefore, on the decoder side, these decimated LFE samples need to be adjusted by the quantization step size and scale factor. After this adjustment, they are used to interpolate the other samples. The choice of the interpolation filter is indicated by LFF as shown in Table 5-14.

AUDIO (Audio data)

The audio data are grouped as nSSC subsubframes, each consisting of 8 samples for each subband. Each sample was quantized by a mid-tread linear quantizer indexed by ABITS. The resultant quantization index may further be encoded by either a Huffman or block code. If it is not, it is included in the bit stream as 2's compliment. All this information is indicated by SEL. The (ABITS,SEL) pair then tells how the subband samples should be extracted from the bit stream (Table 5-26).

The resultant subband samples are then compensated by their respective quantization step sizes and scale factors. Special care shall be paid to possible transient in the subframe. If a transient is flagged by TMODE, one scale factor should be used for samples before the transient and the other one for the after the transient.

For some of the subbands that are ADPCM encoded, the samples of these subbands thus far obtained are actually the difference signals. Their real values shall be recovered through a reverse ADPCM process.

At end of each subsubframe there may be a synchronization check word DSYNC = 0xffff depending on the flag ASPF in the frame header, but there shall be at least a DSYNC at the end of each subframe.

5.7 Unpack Optional Information

The optional information may be included at the end of the frame following completion of the audio data arrays, depending on the status of the optional header flags. This data is not intrinsic to the operation of the decoder but may be used for post processing routines.

Table 5-30: Core optional information

Optional information	Size (bits)
`if ( TIMEF==1 ) // Present only when TIMEF=1.` `  TIMES = ExtractBits(32);`	32

`if ( AUXF==1 ) // Present only if AUXF=1.`
`  AUXCT = ExtractBits(6);`
`else`
`  AUXCT = 0; // Clear it.`
	6

`ByteAlign = ExtractBits (0 ... 7)`	0 to 7
`for (int n=0; n<AUXCT; n++)` `  AUXD[n] = ExtractBits(8);`	8×AUXCT bits
`if ( (CPF==1) && (DYNF!=0) ) )` `  OCRC = ExtractBits(16);`	16

TIMES (Time Code Stamp)

Time code may be used to align audio to video.

AUXCT (Auxiliary Data Byte Count)

The number of auxiliary data bytes to be transmitted in the following AUXD array. It shall be in the range of 1 to 63.

ZeroPadAux (Pad for nSYNCAUX)

The beginning of auxiliary data bytes is aligned on the 32-bit boundary from the beginning of the core stream (DWORD aligned). This is achieved by inserting a necessary number of zero bits after the AUXCT data.

AUXD (Auxiliary Data Bytes)

Unpacking of the auxiliary data is detailed in clause 5.8.2.

OCRC (Optional CRC Check Bytes)

If CPF = 1 then OCRC shall be extracted from the bitstream. The CRC value test shall not be applied.

Additional optional metadata chunks may follow in an arbitrary order. This version of the specification defines one additional metadata chunk denoted by Rev2 Auxiliary Data Chunk. The structure of this chunk is described in clause 5.8.3. The existence of Rev2 Auxiliary Data Chunk is not dependent on the value of the AUXF flag, i.e. the Rev2 Auxiliary Data Chunk may be encoded in the stream even when AUXF=FALSE.

5.8 Optional Information

5.8.1 About Optional Information

This clause describes metadata blocks that may optionally be included in the bitstream.

5.8.2 Auxiliary Data

Table 5-31: Core AUX data

Auxiliary data	Size (bits)

// Advance to DWORD boundary
 	
InputFrame.Advance2NextDWord();
 	
// Extract AUX Sync Word
 	
nSYNCAUX = InputFrame.ExtractBits(32);
	32

// Extract AUX decode time stamp flag
bAUXTimeStampFlag = (ExtractBits(1)==1) ? true : false;
	1

// Extract AUX decode time stamp
if (bAUXTimeStampFlag) {
  InputFrame.Advance2Next4BitPos();
  nMSByte = ExtractBits(8);
  nMarker = ExtractBits(4); // nMaker==1011
  nLSByte28 = ExtractBits(28);
  nMarker = ExtractBits(4); // nMaker==1011
  nAUXTimeStamp = (nMSByte << 28) | nLSByte28;
}
	36

// Extract AUX dynamic downmix flag
bAUXDynamCoeffFlag = (ExtractBits(1)==1) ? true : false;
	1

bInitDwnMixCoeff = false;
DeriveNumDwnMixCodeCoeffs()
{
  unsigned int nPriCh, nNumCoeffs = 0;
  nPriCh = anNumCh[AMODE];
  if (LFF > 0)
  nPriCh++;
  // recall these tables do NOT include a scale row!
  // Check m_nPrmChDownMixType
  switch (m_nPrmChDownMixType)
	3

Auxiliary data	Size (bits)
{	
case DTSDOWNMIXTYPE_1_0:	
nNumCoeffs = nPriCh;	
m_nNumChPrevHierChSet = 1;	
break;	
case DTSDOWNMIXTYPE_LoRo:	
case DTSDOWNMIXTYPE_LtRt:	
nNumCoeffs = 2*nPriCh;	
m_nNumChPrevHierChSet = 2;	
break;	
case DTSDOWNMIXTYPE_3_0:	
case DTSDOWNMIXTYPE_2_1:	
nNumCoeffs = 3*nPriCh;	
m_nNumChPrevHierChSet = 3;	
break;	
case DTSDOWNMIXTYPE_2_2:	
case DTSDOWNMIXTYPE_3_1:	
nNumCoeffs = 4*nPriCh;	
m_nNumChPrevHierChSet = 4;	
break;	
default:	
nNumCoeffs = 0;	
m_nNumChPrevHierChSet = 0;	
break;	
}	
return nNumCoeffs;	
}	
// Unpack the coefficients	
if (bAUXDynamCoeffFlag)	
{	
// Extract the downmix type for primary ch	
nPrmChDownMixType = ExtractBits(3);	
// Extract the downmix code coeffs	
nNumDwnMixCodeCoeffs = DeriveNumDwnMixCodeCoeffs();	
}	
// Extract AUX dynamic downmix coeff codes	
if (bAUXDynamCoeffFlag) {	
if (! ReallocDwnMixCodeCoeff(nNumDwnMixCodeCoeffs))	
return false;	
for (n=0; n < nNumDwnMixCodeCoeffs; n++)	
{	
nTmp = ExtractBits(9);	
panDwnMixCodeCoeffs[n] = nTmp;	
}	
} for (nIndPrmCh=0, n=0; nIndPrmCh<m_nNumChPrevHierChSet; nIndPrmCh++){	
for (nIndXCh=0; nIndXCh<nPriCh; nIndXCh++, n++){	
nTmp = m_panDwnMixCodeCoeffs[n];	
// Extract and test the MSB (NBITSFORDMIXCOEFFWORD-bit words)	variable
// If 1 - > in phase (+1); if 0 -> Out of phase (-1)	
nSign = (nTmp & nMask1) ? 1 : -1;	
nTmp = (nTmp & nMask2);	
if (nTmp>0){	
nTmp--; // -Infinity is not part of the table so decrement index	
if (nTmp>nTblSize)	
return false;	
// convert 24-bit signed coeffs in Q15 to real	
m_panCoreDwnMixCoeffs[n] = (nSign*DmixCoeffTable[nTmp]);	
}	
else	
m_panCoreDwnMixCoeffs[n] = 0.0;	
} }	
ByteAlign = ExtractBits(0 ... 7);	0 to 7
nRev2AUXCRC16 = ExtractBits(16);	16

Navigation to the start location of the auxiliary data is achieved by either reading the AUXCT variable and traversing to the next DWORD or by searching for the DWORD aligned AUX sync word 0x9A1105A0 from the end of the audio frame. Since the data in the auxiliary may be required prior to unpacking the subframe data, the latter approach of searching for the AUX sync word is the suggested method. The auxiliary data may include a 36-bit time-stamp for decode synchronization and any dynamic downmix coefficients.

nSYNCAUX (Auxiliary Sync Word)

The auxiliary sync word is 0x9A1105A0 and is DWORD aligned.

bAUXTimeStampFlag (Auxiliary Decode Time Stamp Flag)

Indicates if a decode time stamp is present in auxiliary data.

nAUXTimeStamp (Auxiliary Decode Time Stamp)

Decode time stamp for synchronizing core audio with another audio stream such as a DTS substream.

The timestamp data is a 36 bit field composed as follows:

$$TimeStamp = (Hours \times Mins \times Sec \times SampleRate) + SampleOffset$$

where:

Hours has range 0 to 23,

Mins has range 0 to 59,

Sec has range 0 to 59,

SampleRate may be 32 000, 44 100 or 48 000 as deduced from the SFREQ table (see Table 5-5),

SampleOffset has range of 0 to 31 999, 44 099 or 47 999.

The timestamp of an encoded frame (*N*) corresponds to that time at the first edge of the first sample period at the start of an audio frame *N* entering into the encoder. The time stamp may also be thought of as a sample counter in which case a value of 0 or other initial value corresponds to the beginning of the first sample period within the first audio frame as processed by the encoder.

bAUXDynamCoeffFlag (Auxiliary Dynamic Downmix Flag)

If this flag is true, it indicates that the down mixing coefficients are included in the stream.

nPrmChDownMixType (Auxiliary Primary Channel Downmix Type)

Designates the dynamic downmix type for the primary channels when bAUXDynamCoeffFlag is true. The downmix type is defined by the parameters in Table 5-32.

Table 5-32: Downmix Channel Groups

nPrmChDownMixType	Downmix primary Channel group to:
000	1/0
001	Lo/Ro
010	Lt/Rt
011	3/0
100	2/1
101	2/2
110	3/1
111	Unused

The downmix coefficients are packed as a NxM table of coefficients. To determine the total number of downmix coefficients packed in the stream, use the nPrmChDownMixType to determine the M (number of resultant downmix channels and the total number of channels encoded to determine N).

panDwnMixCodeCoeffs (Dynamic Downmix Code Coefficients)

Use the nPrmChDownMixType (designated number of fold down channels) and the number of primary channels + LFE channel (if included in audio frame) to determine the number of dynamic downmix code coefficients to extract. Each code coefficient is 9 bits.

See clause D.11 for table lookup to convert encoded downmix coefficient to actual coefficient values.

ByteAlign for nAUXCRC16 (Pad to BYTE Boundary)

This ensures that the nAUXCRC16 field that follows is aligned to a byte boundary to allow fast table-based CRC16 calculation. Append '0's until the bit position is a multiple of 8.

nAUXCRC16 (Auxiliary CRC-16 value)

An auxiliary CRC-16 value is provided to verify both the detection of the auxiliary sync word and the contents of the auxiliary data. Traverse to next byte boundary and read out auxiliary CRC value. This CRC value is calculated for the auxiliary data from positions bAUXTimeStampFlag to the byte prior to the start of the CRC inclusive.

Reserved (Reserved)

This field is reserved for additional auxiliary information. The decoder shall assume that this field is present and of unspecified duration. Therefore in order to continue unpacking the stream, the decoder shall skip over this field using the auxiliary data start pointer and the auxiliary data byte size AUXCT.

5.8.3 Rev2 Auxiliary Data Chunk

5.8.3.1 About the REV2 Aux Data Chunk

The Rev2AUX data chunk contains broadcast metadata such as Dynamic Range Control (DRC) and dialog normalization (dialnorm). Navigation to the start location of the Rev2 auxiliary data chunk is achieved by searching for the DWORD-aligned nSYNCRev2AUX sync word (0x7004C070) by using one of the two following methods:

- searching forward starting after all auxiliary data bytes AUXD are extracted; or
- searching backward starting from the end of the audio frame.

The detected sync word shall be verified by checking the CRC check sum nRev2AUXCRC16. Currently the Rev2 auxiliary data may include an ES metadata flag and a corresponding down-mix attenuation level. In the future, additional metadata may be added in reserved field of the Rev2 Auxiliary Data Chunk.

Padding for nSYNCRev2AUX (ZeroPadRev2Aux)

The beginning of Rev2 Auxiliary Data Chunk is aligned on the 32-bit boundary from the beginning of the core stream (DWORD aligned). This is achieved by inserting a necessary number of zero bits prior to the nSYNCRev2AUX data.

5.8.3.2 Rev2 Auxiliary Data Chunk structure

The Rev2 Auxiliary Data Chunk structure is shown in Table 5-33.

Table 5-33: Rev2 Auxiliary Data Chunk Structure

Rev2 Auxiliary Data Chunk Structure	Size (Bits)
`nSYNCRev2AUX = ExtractBits(32);`	32
`nRev2AUXDataByteSize = ExtractBits(7) + 1;`	7

`if ((nRev2AUXDataByteSize < 3) || (nRev2AUXDataByteSize > 128))`
`{`
`  Error: Invalid range of Rev 2 Auxiliary Data Chunk Size`
`}`	

`bESMetaDataFlag = (ExtractBits(1) == 1) ? TRUE : FALSE;`	1

`if (bESMetaDataFlag==TRUE)`
`{`
`  // Extract Embedded ES Downmix Scale Index`
`  nEmbESDownMixScaleIndex = ExtractBits(8);`
`  // Check index range`
`  if ((nEmbESDownMixScaleIndex < 40) || (nEmbESDownMixScaleIndex >240)){`
`  // Handle error: Invalid Index For a ES Downmix Scaling Parameter`
`  // Look up the scale factor`
`  ESDmixScale = DmixTable[nEmbESDownMixScaleIndex];`
`}`
	8

`if (nRev2AUXDataByteSize>4)`
`  bBroadcastMetadataPresent = (ExtractBits(1) == 1) ? TRUE : FALSE;`
`else`
`  bBroadcastMetadataPresent = FALSE;`
	1

Rev2 Auxiliary Data Chunk Structure	Size (Bits)
If (bBroadcastMetadataPresent == TRUE) {	1
bDRCMetadataPresent = (ExtractBits(1) == 1) ? TRUE : FALSE;	
bDialnormMetadata = (ExtractBits(1) == 1) ? TRUE : FALSE;	1

if (bDRCMetaDataPresent == TRUE) {
// Extract the DRC Version Number
DRCversion_Rev2AUX = ExtractBits(4);
	4

}	

// Byte align to next field
nByteAlign0 = ExtractBits(…);
	1,…,7

// Extract the Rev2AUX DRC values, if present	

if (bDRCMetaDataPresent == TRUE){
// assumes DRCversion_Rev2AUX == 1:
for (subSubFrame=0; subSubFrame < nSSC; subSubFrame++)
subsubFrameDRC_Rev2AUX[subSubFrame] = dts_dynrng_to_db(ExtractBits(8));
	8×subsubframes

}	
// Extract DIALNORM_rev2aux, if present	

if (bDialnormMetadata == TRUE){
DIALNORM_rev2aux = ExtractBits(5);
DNG = - (DIALNORM_rev2Aux);
}
	5

}	
ReservedRev2Aux = ExtractBits(…);	See description
ByteAlignforRev2AuxCRC = ExtractBits(…);	0,…,7
nRev2AUXCRC16 = ExtractBits(16);	16

5.8.3.3 Description of Rev2 Auxiliary Data Chunk fields

nSYNCRev2AUX (Rev2 Auxiliary Data Chunk Sync Word)

The DWORD-aligned Rev2 Auxiliary Data Chunk synchronization word has the value 0x7004C070.

nRev2AUXDataByteSize (Rev2 Auxiliary Data Byte Size)

The nRev2AUXDataByteSize is equal to the size of the Rev2 Auxiliary Data Chunk in bytes from the nRev2AUXDataByteSize to nRev2AUXCRC16 inclusive. This marker also designates the end of the field nRev2AUXCRC16 and allows quick location of the checksum at byte position nRev2AUXDataByteSize - 2 offset from the nRev2AUXDataByteSize inclusive. The nRev2AUXDataByteSize is an unsigned integer with a valid range between 3 and 128 inclusive.

bESMetaDataFlag (ES Metadata Flag)

When the bESMetaDataFlag is TRUE, metadata related to the embedded down-mix from an extended surround (ES) layout is present in the stream, namely nEmbESDownMixScaleIndex which is an index into DmixTable[ ]. When the bESMetaDataFlag is TRUE, it informs the core decoder that the channels encoded in the core stream represent a down-mix from some extended surround (ES) layout with > 5.1 channels. If the bESMetaDataFlag is FALSE or bESMetaDataFlag is not encoded in the stream (i.e. Rev2 Auxiliary Data Chunk), then either no embedded ES downmix metadata exists or should not be used.

nEmbESDownMixScaleIndex (Embedded ES Downmix Scale Index)

This field is encoded in the stream only if the bESMetaDataFlag is TRUE. It corresponds to the amount of attenuation that is applied to the core encoded channels during the process of embedded ES down-mixing. This information is only needed when the core decoded audio represents primary audio that is going to be mixed with the secondary audio. In this case, the mixer needs to obtain this attenuation information from the core decoder in order to adjust the level of secondary audio prior to mixing. The core decoder may choose to combine this scaling with the scaling required by dialog normalization parameter.

The encoded parameter nEmbESDownMixScaleIndex represents an 8-bit index into the DmixTable[ ], which is a scale factors look-up table listed in clause D.11. Although the ESDmixScale parameters are obtained from the DmixTable[ ], their range is limited on the encode side to [-40 dB, 0 dB]. This corresponds to the valid range for nEmbESDownMixScaleIndex to be between 40 and 240 inclusive.

The entries in DmixTable[ ] are unsigned 16-bit integer numbers representing the numbers in column AbsValues[] (from the same table), after multiplication by 2^{15} and rounding to the nearest integer value. The ESDmixScale values are unsigned integer numbers obtained from the entries of DmixTable[ ].

To obtain the actual ES down-mix scale factor (ESDmixScale), the logic shown in Table 5-33 shall be followed.

bBroadcastMetaDataPresent (Broadcast Metadata Present Flag)

When the bBroadcastMetaDataPresent flag is TRUE, metadata related to the application of DRC and Dialnorm are both present in the Rev2AUX chunk. Therefore, bDRCMetaDataPresent and bDialnormMetadataPresent flags shall be checked and any associated metadata shall be applied to the core stream channels.

Presence of the bBroadcastMetaDataPresent flag and the associated metadata may NOT be guaranteed even when the Rev2AUX data chunk is present in the stream. Decoders that use this metadata SHALL determine the presence of the bBroadcastMetaDataPresent flag based on the nRev2AUXDataByteSize. In particular the bBroadcastMetaDataPresent flag is present in the stream if and only if the nRev2AUXDataByteSize > 4.

bDRCMetaDataPresent (DRC Metadata Present Flag)

This flag will be present if bBroadcastMetaDataPresent flag is TRUE. When the bDRCMetaDataPresent flag is TRUE, metadata related to the application of DRC will be present in the Rev2AUX chunk and the DRC values in the Rev2AUX data chunk should be used instead of any dynamic range control coefficients found in the legacy core stream (indicated by flag DYNF). In addition, channels encoded in the core stream should apply the subsequent DRC values to subsubframes. If the bBroadcastMetaDataPresent flag is FALSE, no subsubframe DRC values are present in the Rev2AUX data chunk.

bDialnormMetaDataPresent (Dialnorm Metadata Present Flag)

This flag will be present if bBroadcastMetaDataPresent flag is TRUE. When the bDialnormMetaDataPresent flag is TRUE, metadata related to the application of dialnorm will be present in the Rev2AUX chunk and the dialog normalization values in the Rev2AUX data chunk should be used instead of the DIALNORM field found in the legacy core stream. In addition, the Rev2AUX dialnorm value should be smoothed and applied to the channels encoded in the core stream. If the bBroadcastMetaDataPresent flag is FALSE, no dialnorm values are present in the Rev2Aux data chunk.

DRCversion_Rev2AUX (DRC Version)

This field will be present only if bBroadcastMetaDataPresent flag is TRUE and if bDRCMetaDataPresent flag is TRUE. DRCversion_Rev2AUX is a four bit field which is used to determine the version of DRC algorithm which the encoder used. The first version starts at 0x1. Decoders will support DRC version 0x1 up to the latest version which they support. Currently only DRCversion_Rev2AUX = 1 is supported in the Rev2Aux chunk. If the encoder is supplying DRC information with a version number higher than that which is supported by the decoder, the supplied Rev2Aux DRC values should be ignored and no DRC should be applied.

nByteAlign0 (ByteAlignvaries)

This 0-bit padding ensures that the field that follows is aligned to a byte boundary. Append '0' bits until the bit position is a multiple of 8. This field will be 1 bit if the DRCversion_Rev2AUX field was present and will be 5 bits if the DRCversion_Rev2AUX field was not present.

subsubFrameDRC_Rev2AUX[ ] (DRC Values)

This field will be present only if bBroadcastMetaDataPresent flag is TRUE and if bDRCMetaDataPresent flag is TRUE. Each subsubFrameDRC_Rev2AUX[ ] field falls on a byte boundary. Currently only DRC version 1 is supported, which is single band mode. In single band mode, one 8 bit value is transmitted for each subsubframe, as detailed in Table 5-34.

Table 5-34: Number of Rev2AUX DRC bytes transmitted per frame length

Frame Length	DRC bytes (1 per subsubframe)
512	2
1 024	4
2 048	8

In the CA stream, each subsubframe is 256 samples in length, corresponding to 5,33 ms at 48 000 samples per second (5,33 ms = subsubframe length in sample/ sampling rate). Table 5-35 shows the relationship between the frame sizes and number of bits used for DRC values:

Table 5-35: Rev2AUX DRC bits per frame

Frame length in samples	Number of subsubframes per frame	Frames/sec	No. of bits used for DRC values per frame	Bits/sec for DRC values only	DRC extension overhead (bits/frame)	DRC extension total not including zero padding (bits/frame)	DRC extension total not including zero padding (bits/sec)
512	2	9 375	16	1 500	10	26	2 438
1 024	4	46 875	32	1 500	10	42	1 969
2 048	8	234 375	64	1 500	10	74	1 808

Each DRC byte value is extracted from the bitstream and converted into a dB gain by function `dts_dynrng_to_db()`.

DIALNORM_rev2aux (Dialog Normalization Parameter)

The DIALNORM_rev2aux field will be present only if bBroadcastMetaDataPresent flag is TRUE and if bDialnormMetaDataPresent flag is TRUE. Field DIALNORM_rev2aux falls on a byte boundary. If the encoded stream contains both a DIALNORM field and a DIALNORM_rev2aux field, DIALNORM_rev2aux takes priority. DIALNORM_rev2aux is a 5-bit field which is used to determine the dialog normalization parameter.

The dialog normalization gain (DNG), in dB, is specified by the encoder operator and is used to directly scale the decoder output samples. In the DTS stream, the information about the DNG value is transmitted as described in Table 5-36.

Table 5-36: Rev2AUX Dialog Normalization Parameter

Dialog Normalization Gain (DNG) Applied to the Decoder Outputs [dB]	DIALNORM_rev2aux (binary)	DIALNORM_rev2aux (unsigned int)
0	0b00000	0
-1	0b00001	1
-2	0b00010	2
-3	0b00011	3
-4	0b00100	4
-5	0b00101	5
-6	0b00110	6
-7	0b00111	7
-8	0b01000	8
-9	0b01001	9
-10	0b01010	10
-11	0b01011	11
-12	0b01100	12
-13	0b01101	13
-14	0b01110	14
-15	0b01111	15
-16	0b10000	16
-17	0b10001	17
-18	0b10010	18
-19	0b10011	19
-20	0b10100	20
-21	0b10101	21
-22	0b10110	22
-23	0b10111	23
-24	0b11000	24

Dialog Normalization Gain (DNG) Applied to the Decoder Outputs [dB]	DIALNORM_rev2aux (binary)	DIALNORM_rev2aux (unsigned int)
-25	0b11001	25
-26	0b11010	26
-27	0b11011	27
-28	0b11100	28
-29	0b11101	29
-30	0b11110	30
-31	0b11111	31

ReservedRev2Aux (Reserved bits)

This field is reserved for additional metadata information that may be added in the future. The decoder shall assume that this field is present and of unspecified duration. Therefore, in order to continue unpacking the stream, the decoder shall skip over this field skipping nRev2AUXDataByteSize bytes from the bESMetaDataFlag inclusive.

ByteAlignforRev2AuxCRC (Pad for nRev2AUXCRC16)

This zero-padding field ensures that the nRev2AUXCRC16 field that follows is aligned to a byte boundary to allow fast table-based CRC16 calculation. Append '0' bits until the bit position is a multiple of 8.

nRev2AUXCRC16 (Rev2 Auxiliary CRC-16 value)

A Rev2 auxiliary CRC-16 value is provided to verify both the detection of the Rev2 auxiliary sync word and the contents of Rev2 Auxiliary Data Chunk. To locate the position of the nRev2AUXCRC16 data field, start from the beginning position of field nRev2AUXDataByteSize and jump forward (nRev2AUXDataByteSize - 2) bytes. This CRC value is calculated for the Rev2 Auxiliary Data Chunk from the position of nRev2AUXDataByteSize to the ByteAlignforRev2AuxCRC, inclusive.

6 Core Extensions

6.1 About the Core Extensions

The generalized concept of core + extension coding is well established in the context of DTS encoding. The present document describes the extensions (components) found in any DTS stream. These components include:

- XCH - Extra centre surround (Cs) channel with 6.1->5.1 down-mix embedded in the core stream using default down-mix coefficients (Ls^{dm} = Ls+0,7071Cs; Rs^{dm} = Rs+0,7071Cs).
- X96 - High frequency components introduced by higher sampling rates (88,2/96 kHz).
- XBR - Extended resolution for the channels encoded in the core sub-stream (requires bit-rates > 1,5 Mbps but guarantees backward compatibility of a 1,5 Mbps core sub-stream).
- XXCH - Extra channels beyond 5.1.

Some of these components may exist in either the core sub-stream or the extension sub-stream. The extension substream is defined in clause 7.

6.2 X96 Extension

6.2.1 About the X96 Extension

The generalized concept of core+96 kHz-extension coding is illustrated in Figure 6-1. To encode 96 kHz LPCM, the input audio stream is fed to a 96 to 48 kHz down sampler and the resulting 48 kHz signal is encoded using a standard core encoder as in Figure 6-1, Section A, as follows:

- In the "Preprocess Input Audio" block, the original 96 kHz/24-bit LPCM audio is first delayed and then passed through the extension 64-band analysis filter bank. Signal "1" in this case consists of the extension subband samples at 96 kHz/64.
- The core data consists of the core audio codes in 32 subbands. In the "Reconstruct Core Audio Components" block, the core audio codes are inversely quantized to produce the reconstructed core subband samples at 48 kHz/32. These subband samples correspond to signal "2".
- In the "Generate Residuals" block, the reconstructed core subband samples are subtracted from the extension subband samples in the lower 32 subbands. The extension subband samples in the upper 32 bands remain unaltered. These residual subband samples in the 64 bands correspond to signal "3".
- The "Generate Extension Data" block processes the residual subband samples and generates the extension data that, along with the core data, is assembled in a packer to produce a core + extension bitstream.

In the 96 kHz decoder, as in Figure 6-1, Section B, the unpacker first separates the core + extension stream into the core and extension data. The core subband decoder, in the Reconstruct Core Audio Components block, processes the core data and produces the reconstructed core subband samples (same as signal "2" generated in the encoder). Next, in the Reconstruct Residual Components block, the extension subband decoder uses the extension data to generate the reconstructed residual subband samples in the 64 bands. In the Recombine Core and Residual Components block the core subband samples are added to the lower 32 bands of residual subband samples to produce the extension subband samples in the 64 bands. In the same block, the synthesis 64-band filter bank processes the extension subband samples and generates the 96 kHz 24-bit LPCM audio. The combining of reconstructed residuals and core signals on the decoder side, as in Figure 6-1, Section B, is also done in subband domain.

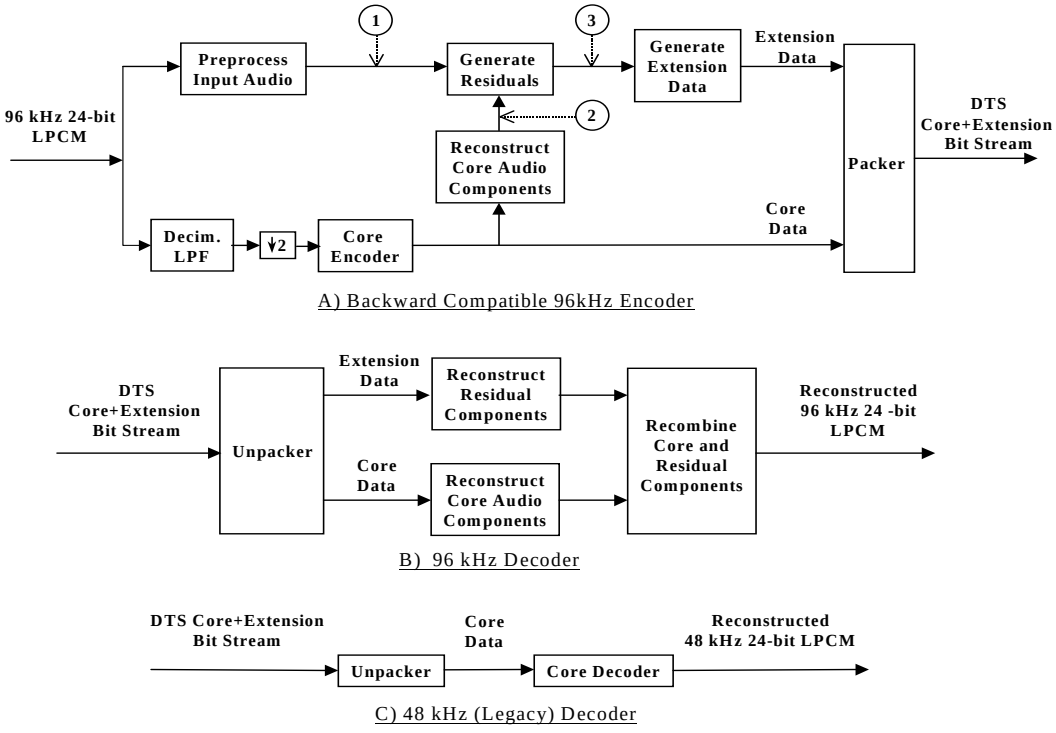


Figure 6-1: The Concept of Core + Extension Coding Methodology

When a 48 kHz-only (legacy) decoder is fed the core + extension bitstream, as in Figure 6-1, Section C, the extension data fields are ignored and only the core data is decoded. This results in 48 kHz core LPCM audio output.

6.2.2 DTS Core + 96 kHz-Extension Encoder

The block diagram in Figure 6-2 shows the main components of the encoding algorithm. The input digital audio signal with a sampling frequency up to 96 kHz and a word length up to 24 bits is processed in the core branch and extension branch. In the core branch input audio is low-pass filtered (LPF) to reduce its bandwidth to below 24 kHz and then decimated by a factor of two, resulting in a 48 kHz sampled audio signal. The purpose of this LPF decimation is to remove signal components that cannot be represented by the core algorithm. The down sampled audio signal is processed in a 32-band analysis cosine modulated filter bank (QMF) that produces the core subband samples. The core bit allocation routine based on the energy contained in each of the subbands and configuration of the core encoder determines the desired quantization scheme for each of the subbands. The core subband encoder performs quantization and encoding after which the audio codes and side information are delivered to the packer. The packer assembles this data into a core bitstream. The X96 extension can either be an extension of the core audio frame or included with a DTS substream.

In the extension branch the delayed version of input audio is processed in a 64-band analysis cosine modulated filter bank (QMF) that produces the extension subband samples. Inverse quantization of the core audio codes produces the reconstructed core subband samples. Subtracting these samples from the extension subband samples in the lower 32 bands generates the residual subband samples. The residual signals in the upper 32 subbands are unaltered extension subband samples in corresponding bands. The delay of input audio is such that reconstructed core subband samples and extension subband samples in the lower 32 bands are time-aligned before the residual signals are produced i.e.:

$$\text{Delay} = \text{Delay}_{\text{DecimationLPF}} + \text{Delay}_{\text{CoreQMF}} - \text{Delay}_{\text{ExtensionQMF}}.$$

The extension bit allocation routine based on the energy of residuals in each of the subbands and configuration of the extension encoder determines the desired quantization scheme for each of 64 subbands. The residual samples in subbands are encoded using a multitude of adaptive prediction, scalar/vector quantization and/or Huffman coding to produce the residual codes and extension side information. The packer assembles this data into an extension bitstream.

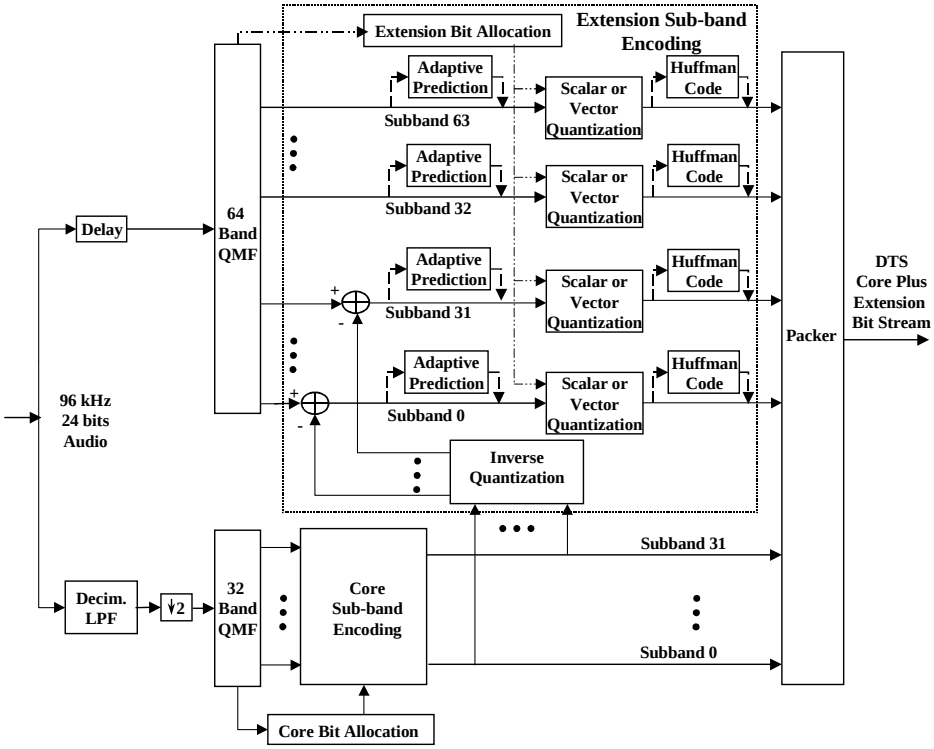


Figure 6-2: The Block Diagram of DTS Core + Extension Encoder

6.2.3 DTS Core + 96 kHz Extension Decoder

On the decoder side, the core and extension parts of the encoded bitstream are fed to their respective subband decoders. The reconstructed core subband samples are added to the corresponding residual subband samples in lower 32 bands. The reconstructed residual subband samples in the upper 32 bands remain unaltered. Passing the resulting extension subband samples through the synthesis 64-band QMF filter bank produces the 96 kHz sampled PCM audio. Figure 6-3 shows the block diagram of the core + extension decoder.

If the encoded bit-stream does not contain the extension data, the decoder, based on its hardware configuration, uses:

- a 32-band QMF with core subband samples as inputs to synthesize the 48 kHz sampled PCM audio;
- a 64-band QMF with inputs being core subband samples in the lower 32 bands and "zero" samples in the upper 32 bands to synthesize the interpolated PCM audio sampled at 96 kHz.

The existing DTS core decoders when receiving the core + extension bitstream will extract and decode the core data to produce the 48 kHz sampled PCM audio. The decoder ignores the extension data by skipping the extraction until the next DTS synchronization word.

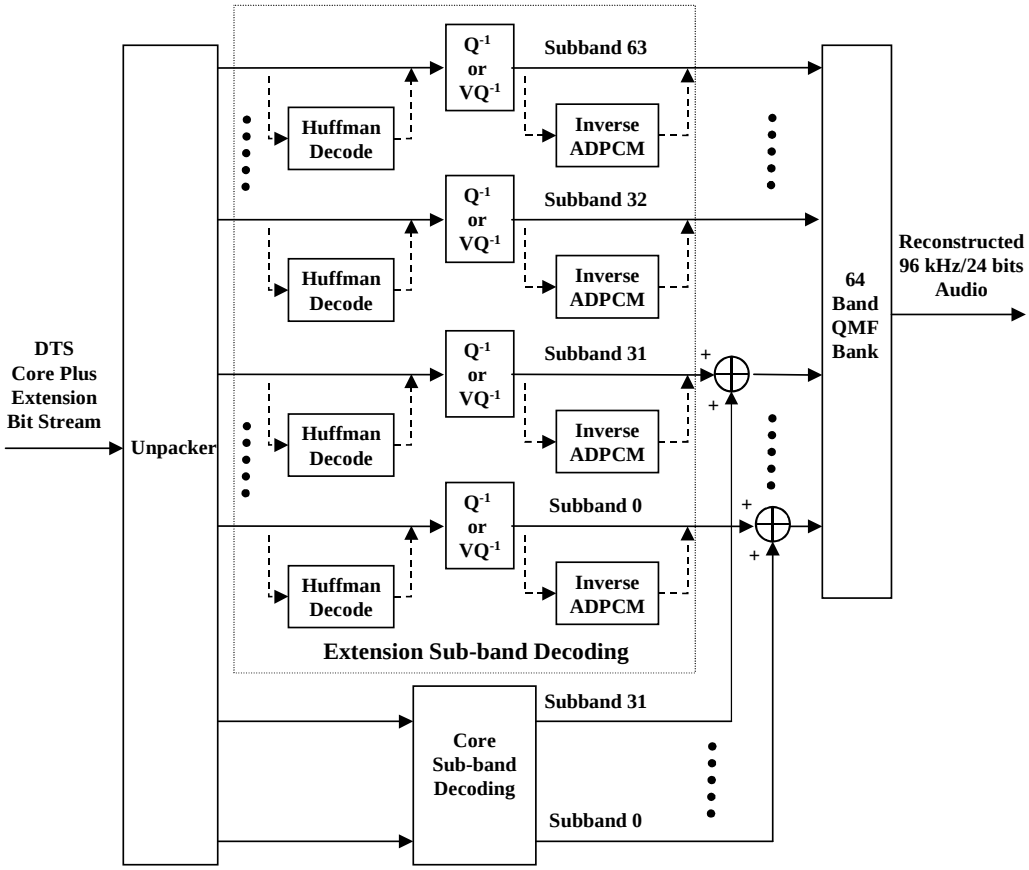


Figure 6-3: The Block Diagram of DTS Core + extension Decoder

6.2.4 Extension (X96) Bitstream Components

6.2.4.1 About the X96 Bitstream Components

The X96 extension may be a part of the core substream (denoted by the DTS_BCCORE_X96**)** or a part of the extension substream (denoted by the DTS_EXSUB_STREAM_X96). During the synchronization procedure, the decoder will determine whether the DTS_BCCORE_X96 or DTS_EXSUB_STREAM_X96 is being decoded.

The frame of X96 data is divided into a frame header and up to four channel sets. The frame header structure is different for the case of DTS_BCCORE_X96 extension and the case of DTS_EXSUB_STREAM_X96 extension. In case of the DTS_BCCORE_X96 only one channel set may exist in the X96 stream.

Each channel set has its own channel set header as shown in Figure 6-4. In the case of DTS_EXSUB_STREAM_X96, the Cycle Redundancy Check (CRC) word is included at the end of frame header to allow detection of errors in the frame header data. In addition, the CRC words may be included at the end of each channel set header to allow detection of errors in the channel set header data.

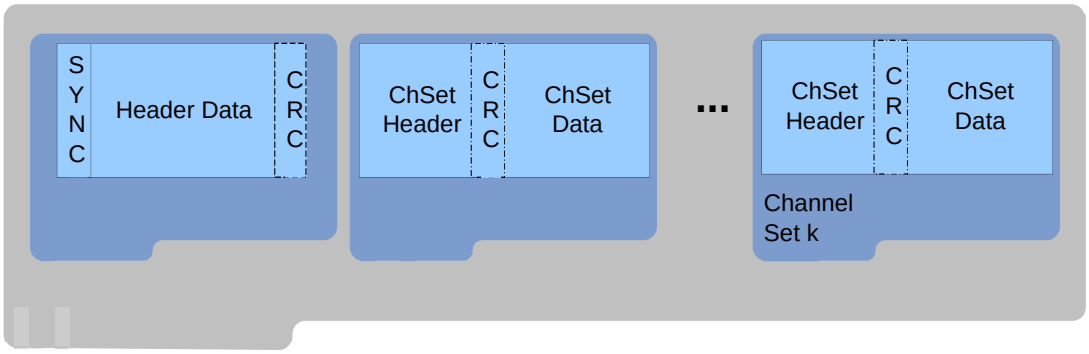


Figure 6-4: X96 data frame structure

The channel set data is subdivided into the subframes. Each subframe consists of a subframe header and the audio data. The audio data is organized in subsubframes as shown in Figure 6-5. The number of subframes is the same for all channel sets and is equal to the number of subframes in the core frame. Similarly, the number of subsubframes is the same for all subframes of all channel sets and is equal to the number of subsubframes within each subframe of the core frame. In other words, the subframe and subsubframe partitioning within the X96 frame follows the partitioning present in the core frame.

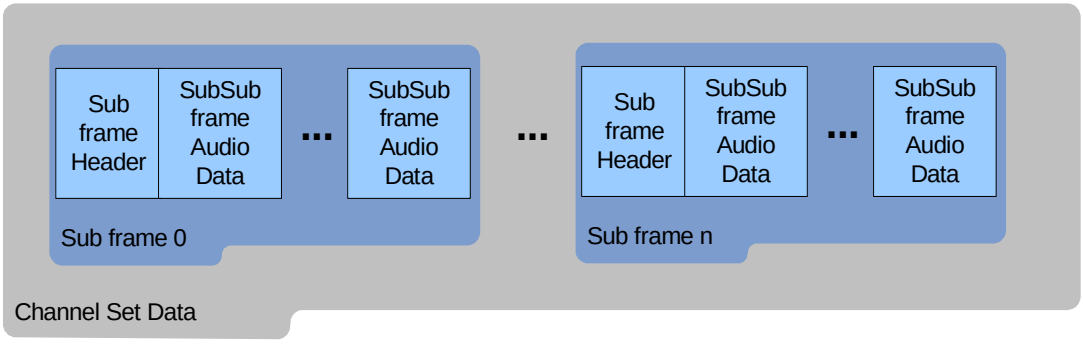


Figure 6-5: Channel Set Data Structure

6.2.4.2 DTS_BCCORE_X96 Frame Header

This clause describes the X96 extension when it is present in the core extension of the core extension substream.

Table 6-1: DTS_ BCCORE _X96 Frame Header Structure

DTS_ BCCORE _X96 Frame Header Syntax	Size (Bits)
SYNC96= ExtractBits(32);	32
FSIZE96 = ExtractBits(12)+1;	12
REVNO = ExtractBits(4);	4

SYNC96 (DTS_BCCORE_X96 Extension Sync Word)

The synchronization word SYNC96 = 0x1D95F262 for the DTS_BCCORE_X96 extension data comes after the core audio data within the core substream. For 16-bitstreams the sync word is aligned to 32-bit word boundary. In the case of 14-bitstreams SYNC96 is aligned to both 32-bit and 28-bit word boundaries meaning that 28 MSBs of the SYNC96 appear as 0x07651F26.

To reduce the probability of false synchronization caused by the presence of pseudo sync words, it is imperative to check the distance between the detected sync word and the end of current frame (as indicated by FSIZE). This distance in bytes shall match the value of FSIZE96 (see below).

After the synchronization is established a flag nX96Present is set and the output sampling frequency is selected as:

```
OutSamplingFreq = SFREQ;
    if ( nX96Present)
OutSamplingFreq = 2*OutSamplingFreq;
```

NOTE: SFREQ corresponds to a sampling frequency of reconstructed audio in the core decoder.

FSIZE96 (DTS_BCCORE_X96 Extension Frame Byte Data Size)

FSIZE96 is the byte size of DTS_ BCCORE _X96 extension data. Valid range for FSIZE96: 96 - 4 096.

REVNO (Revision Number)

The revision number for the high-frequency extension processing algorithm.

Table 6-2: REVNO

REVNO	Frequency Extension Encoder Software Revision Number
0	Reserved
1	Currently in use (compatible with the Rev1.0 specification)
2 - 7	Future revision (compatible with the Rev1.0 specification)
8	Currently in use (incompatible with the original Rev1.0 specification)
9 - 15	Future revision (incompatible with the original Rev1.0 specification)

Decoders designed using this Rev3.0 specification shall be able to decode all streams when REVNO < 9. If decoder is not compatible with some algorithm revisions (REVNO > 8), it shall ignore the DTS_ BCCORE_X96 extension stream.

6.2.4.3 DTS_EXSUB_STREAM_X96 Frame Header

This clause describes the X96 extension when it is present in the extension substream of the DTS-HD stream.

Table 6-3: DTS_EXSUB_STREAM_X96 Frame Header Structure

DTS_EXSUB_STREAM_X96 Frame Header Syntax	Size (Bits)
`SYNCX96 = ExtractBits(32);`	32
`nHeaderSizeX96 = ExtractBits(6)+1;`	6
`REVNO = ExtractBits(4);`	4
`bCRCPresent4ChSetHeaderX96 = ExtractBits(1);`	1
`nNumChSetsInX96 =  ExtractBits(2)+1;`	2
`for (nChSet=0; nChSet < nNumChSetsInX96; nChSet ++)` `  pnuChSetFsizeX96[nChSet] = ExtractBits(12)+1;`	12
`for (nChSet=0; nChSet < nNumChSetsInX96; nChSet ++)` `  nuChInChSetX96[nChSet] = ExtractBits(3) + 1;`	3
`ReservedHeaderX96 = ExtractBits(…);`	…
`ByteAlignHeaderX96 = ExtractBits(0 … 7);`	0...7
`nCRC16HeaderX96 = ExtractBits(16);`	16

SYNCX96 (DTS_EXSUB_STREAM_X96 Sync Word)

The DWORD aligned DTS_ EXSUB_STREAM _X96 synchronization word has a value SYNCX96 = 0x1D95F262. During sync detection, the nCRC16HeaderX96 checksum is used to further verify that the detected sync pattern is not a random alias. After the synchronization is established, the flag (nX96Present) is set and the output sampling frequency is selected as:

```
OutSamplingFreq = SFREQ;
if ( nX96Present)
```

NOTE: SFREQ corresponds to a sampling frequency of reconstructed audio in the core decoder.

nHeaderSizeX96 (DTS_EXSUB_STREAM_X96 frame header length)

The size of the header in bytes from SYNCX96 to nCRC16HeaderX96 inclusive. This value determines the location of the first channel set header. This marker also designates the end of the field, nCRC16HeaderX96 and allows quick location of the checksum at byte position nHeaderSizeX96-2.

REVNO (Revision Number)

The revision number for the high frequency extension processing algorithm has the same coding and interpretation as described in Table 6-2.

bCRCPresent4ChSetHeaderX96 (CRC presence flag for channel set header)

When bCRCPresent4ChSetHeaderX96 = true the 16-bit CRC word for the channel set header is present at the end of each channel set header. For the case of DTS_BCCORE_X96 the default value for the bCRCPresent4ChSetHeaderX96 is false.

nNumChSetsInX96 (Number of Channel Sets)

All channels within the DTS_EXSUB_STREAM_X96 extension are organized into individually decodable channel sets. The nNumChSetaInX96 is the number of channel sets that are present in X96 component.

pnuChSetFsizeX9 (Channel Set Data Byte Size)

The pnuChSetFsizeX96[nChSet] indicates the total number of data bytes in each nChSet channel set of the DTS_EXSUB_STREAM_X96 frame. Starting from the SYNCX96 and using the cumulative sum of nHeaderSizeX96 and pnuChSetFsizeX96[k] (over all channel sets k=0, … nChSet -1) as offset, the decoder may traverse to the beginning of channel set header data in the channel set nChSet.

nuChInChSetX9 (Number of channels in a channel set)

Indicates the number of channels in the channel set.

ReservedHeaderX96 (Reserved)

This field is reserved for additional DTS_EXSUB_STREAM_X96 header information. The decoder shall assume that this field is present and of unspecified length. Therefore, in order to continue unpacking the stream, the decoder shall skip over this field using the DTS_EXSUB_STREAM_X96 header start pointer and the DTS_EXSUB_STREAM_X96 header size nHeaderSizeX96.

ByteAlignHeaderX96 (Pad to BYTE boundary)

This field ensures that the CRC16 field that follows is aligned to a byte boundary to allow fast table based CRC16 calculation. Append '0's until bit position is a multiple of 8.

nCRC16HeaderX96 (CRC16 of X96 frame header)

This field represents the 16-bit CRC check word of the entire DTS_EXSUB_STREAM_X96 header from position nHeaderSizeX96 to ByteAlignHeaderX96 inclusive.

6.2.4.4 X96 Channel Set Header

Table 6-4: X96 Channel Set Header Structure

X96 Channel Set Header Syntax	Size (Bits)
if (m_nuCoreExtensionMask & DTS_EXSUB_STREAM_X96) nuChSetHeaderSizeX96 = ExtractBits(7)+1;	7
HIGHRES-FLAG96K = ExtractBits(1);	1

if (REVNO<8){
  nSBS96 = ExtractBits(5);
  if (nSBS96<0 || nSBS96>27)
  return SYNC_ERROR_DEF;
}
else
  nSBS96=32;
	5

// For DTS_EXSUB_STREAM_X96: nPCHS=nuChInChSetX96[ChSet];
// For DTS_BCCore_X96 nPCHS is obtained from the core
for (ch=0; ch<nPCHS; ch++){
  SBE96[ch] = ExtractBits(6);
  anSBE96[ch] = SBE96[ch] + 1;
}
	6

for (ch=0; ch<nPCHS; ch++) JOINX96[ch] = ExtractBits(3);	3
for (ch=0; ch<nPCHS; ch++) SHUFF96[ch] = ExtractBits(3);	3
for (ch=0; ch<nPCHS; ch++) BHUFF96[ch] = ExtractBits(3);	3

// ABITS96=2:
n=0;
for (ch=0; ch<nPCHS; ch++)
  SEL96[ch][n] = ExtractBits(1);
// ABITS96 = 3 to 6:
for (n=1; n<5; n++){
  for (ch=0; ch<nPCHS; ch++)
  SEL96[ch][n] = ExtractBits(2);
}
// ABITS96 = 7
for (ch=0; ch<nPCHS; ch++)
  SEL96[ch][n] = ExtractBits(3);
 	
if (HIGHRESFLAG96K == 1){
  // ABITS96 = 8 to 11:
  for (n=5; n<10; n++){
  for (ch=0; ch<nPCHS; ch++)
  SEL96[ch][n] = ExtractBits(3);
  }
  // ABITS96 = 12 to 15:
  for (n=10; n<16; n++){
  for (ch=0; ch<nPCHS; ch++)
  SEL96[ch][n] = 0; // Not transmitted, set to 0
  }
}
	Variable bits

if (m_nuCoreExtensionMask & DTS_EXSUB_STREAM_X96){
  ReservedHeaderChSet = ExtractBits(…);
  ByteAlignHeaderChSet = ExtractBits(0 … 7);
  If (bCRCPresent4ChSetHeaderX96==true)
  nCRC16CheSetHeader = ExtractBits(16);
else{
  if (CPF==1)
  AHCRC96 = ExtractBits(16);
}
	…
	0...7
	16
	16

NuChSetHeaderSizeX96 (Channel set header length)

This field is present only for the case of DTS_EXSUB_STREAM_X96 stream. The size of the channel set header in bytes from the nuChSetHeaderSizeX96 to either ByteAlignHeaderChSet (when bCRCPresent4ChSetHeaderX96 = false) or nCRC16HeaderChSet (when bCRCPresent4ChSetHeaderX96 = true), inclusive. This value determines the beginning of the channel set audio data. If the bCRCPresent4ChSetHeaderX96 = true, this marker also designates the end of the field nCRC16HeaderChSet and allows quick location of the checksum at byte position nuChSetHeaderSizeX96 - 2.

HIGHRESFLAG96K (High Resolution Flag)

The high resolution flag determines the upper limit on ABITS96 and consequently, the maximum number of quantization levels for the audio samples as described in Table 6-5.

Table 6-5: High Resolution Flag

HIGHRESFLAG96K	ABIT
0	0 - 7
1	0 - 15

nSBS96 (First encoded subband transmitted only if REVNO<8)

This field indicates the first active extension subband in each primary channel. Extension subband samples in subbands [0 : (nSBS96-1)] are assumed to be zeros.

SBE96 and anSBE96 (Last encoded subband)

This field indicates the last subband in the audio channel, ch, that is encoded without the use of joint intensity coding ($31 \leq SBE96[ch] \leq 63$). When the joint intensity coding in the audio channel, ch, is disabled, the extension subband samples in subbands above SBE96[ch] are assumed to be zeros (anSBE96 is used in the extension substream processing).

JOINX96 (Joint Intensity Coding Index)

This field indicates if joint intensity coding is enabled for channel, ch and which audio channel is the source channel from which channel ch will copy subband samples. The construction of JOINX96 is done according to Table 5-22.

SHUFF96 (Scale Factor Encoder Select)

The scale factors of a channel are quantized nonlinearly using 6-bit (64-level, 2,2 dB per step) square root table. The quantization indices may be further compressed by one of the five Huffman codes (129 levels) and this information is transmitted to the decoder by SHUFF96[ch]. Scale factors are differentially encoded prior to the Huffman encoding.

Table 6-6: Scale Factor Encoder Select SHUFF96

SHUFF96	Code Book	Square-root Table
0	SA129	6 bit (clause D.1.1)
1	SB129	6 bit (clause D.1.1)
2	SC129	6 bit (clause D.1.1)
3	SD129	6 bit (clause D.1.1)
4	SE129	6 bit (clause D.1.1)
5	linear	6 bit (clause D.1.1)

BHUFF96 (Bit Allocation Quantizer Select)

This field indicates which codebook should be used to encode the bit allocation index ABITS96. The bit allocation indices may be further compressed by one of the seven Huffman codes. This information is transmitted to the decoder by BHUFF96[ch]. When Huffman encoding is used the bit allocation indices are first differentially encoded.

Table 6-7: Bit Allocation Encoder Select BHUFF96

BHUFF96	Code Book for HIGHRESFLAG96K=0	Code Book for HIGHRESFLAG96K=1
0	A17	A33
1	B17	B33
2	C17	C33
3	D17	D33
4	E17	E33
5	F17	F33
6	G17	G33
7	3-bit linear	4-bit linear

SEL96 (Quantization Index Codebook Select)

After subband samples are quantized using a mid-tread linear quantizer, the quantization indices are further encoded using entropy (Huffman) coding in order to reduce the bit rate. For channel ch, the selection of a particular codebook for a mid-tread linear quantizer indexed by ABITS96[ch] is transmitted to the decoder as SEL96[ch][ABITS96[ch]-2]. Table 6-8 depicts the quantization index codebook values.

Table 6-8: Quantization Index Codebook Select SEL96

ABIT	Quantization Type	# bits for SEL	Quantization Index Code-book Select SEL							
			0	1	2	3	4	5	6	7
0	No Bits Allocated	0	Not Transmitted							
1	16 element VQ	0	VQ16							
2	3-level SQ	1	A3	V3						
3	5-level SQ	2	A5	B5	C5	V5				
4	7-level SQ	2	A7	B7	C7	V7				
5	9-level SQ	2	A9	B9	C9	V9				
6	13-level SQ	2	A13	B13	C13	V13				
7	17-level SQ	3	A17	B17	C17	D17	E17	F17	G17	V17
In case of HIGHRESFLAG96K = 1 ABIT takes values from 0 to 15										
8	25-level SQ	3	A25	B25	C25	D25	E25	F25	G25	V25
9	33 or 32 level SQ	3	A33	B33	C33	D33	E33	F33	G33	NFE
10	65 or 64 level SQ	3	A65	B65	C65	D65	E65	F65	G65	NFE
11	129 or 128 level SQ	3	A129	B129	C129	D129	E129	F129	G129	NFE
12	256-level SQ	0	NFE							
13	512-level SQ	0	NFE							
14	1 024-level SQ	0	NFE							
15	2 048-level SQ	0	NFE							
NOTE: NFE = No further encoding is used to encode the linearly quantized subband samples. A,B,C,D,E,F,G = Subband samples are encoded by Huffman code. V = 4 subband samples are grouped and encoded using 4-element block code.										

ReservedHeaderChSet (Reserved)

This field is present only for the case of DTS_EXSUB_STREAM_X96 stream. This field is reserved for additional channel set header information. The decoder shall assume that this field is present and of unspecified duration. Therefore in order to continue unpacking the stream, the decoder shall skip over this field using the channel set header start pointer and the channel set header size nuChSetHeaderSizeX96.

ByteAlignHeaderChSet (Pad to BYTE boundary)

This field is present only for the case of the DTS_EXSUB_STREAM_X96 stream. This field ensures that the CRC16 field that follows is aligned to a byte boundary to allow fast table based CRC16 calculation. Append '0's until bit position is a multiple of 8.

nCRC16CheSetHeader (Channel Set Header CRC Check Word)

This field is present only for the case of the DTS_EXSUB_STREAM_X96 stream and only when the bCRCPresent4ChSetHeaderX96 is true. It checks if there is any error in the bitstream from the beginning of channel set header data up to this point.

AHCRC96 (Audio Header CRC Check Word)

This field is present only for the case of the DTS_ BCCORE _X96 and only when the core parameter CPF is 1. If this field is present, the value shall be extracted from the bitstream. This value will not be used.

Unpack 96 kHz Extension Subframes

Table 6-9: X96 Channel Subframe Processing

X96 Extension Subframes	Size (Bits)

for (ch=0; ch<nPCHS; ch++){
  for (n=nSBS96; n<anSBE96[ch]; n++){
  PMODE96[ch][n] = ExtractBits(1);
  }
}
for (ch=0; ch<nPCHS; ch++){
  for (n=nSBS96; n<anSBE96[ch]; n++){
  if (PMODE96[ch][n]>0) {
  // Extract the PVQs
  nVQIndex = ExtractBits(12);
  // Look-up 4 ADPCM coefficients
  ADPCMCoeffVQ.LookUp(nVQIndex,raADPCMcoeff[ch][n], 4)
  }
  }
}
	1 bit per active subband
	
	12 bits per occurrence

for (ch=0; ch<nPCHS; ch++) {
  nQSelect = BHUFF96[ch];
  // Undo differential encoding
  QABITS96->ppQ[nQSelect]->ClearDeltaSum();
 	
for (n=nSBS96; n<anSBE96[ch]; n++)
  QABITS96->ppQ[nQSelect]->InverseQ(InputFrame, ABITS96[ch][n])
}
	Variable bits per each active subband

for (ch=0; ch<nPCHS; ch++) {
  // Reset SCALES
  for (n=0; n<NumSubband; n++) {
  SCALES96[ch][n] = 0;
  }
  // Select RMS table
  pScaleTable = &RMS6Bit;
 	
  // Select quantizer
  nQSelect = SHUFF96[ch];
 	
  // Clear differential accumulation.
  QSCALES.ppQ[nQSelect]->ClearDeltaSum();
  for (n=nSBS96; n<anSBE96[ch]; n++) {
  // Scale factor index
  QSCALES.ppQ[nQSelect]->InverseQ(InputFrame, nScale);
  // RMS look up
  pScaleTable->LookUp(nScale, SCALES96[ch][n]);
  }
}
	Variable bits per each active subband

for (ch=0; ch<nPCHS; ch++)
  if (JOINX96[ch]>0)
  JOIN_SHUFF96[ch] = ExtractBits(3);
for (ch=0; ch<nPCHS; ch++){
  if (JOINX96[ch]>0) {
  // Get source channel.
  nSourceCh = JOINX96[ch]-1;
  // Select quantizer.
  nQSelect = JOIN_SHUFF96[ch];
  for (n=anSBE96[ch]; n<anSBE96[nSourceCh]; n++) {
  // Extract joint scale factors
  pQJOIN_SCALES->ppQ[nQSelect]->InverseQ(InputFrame, nJscale);
  // Biased by midpoint
  nJscale += 64;
  // Look up scale factor of joint intensity coding
  JScaleTbl.LookUp(nJscale,JOIN_SCALES96[ch][n]);
  }
  }
}
	3 bits per ch. Assuming JOINX96[ch]>0
	
	Variable bits

if (CPF == 1) SICRC96 = ExtractBits(16);	16 bits (if CPF=1)

6.2.4.5 96 kHz Extension Side Information

PMODE96

PMODE96 Indicates if ADPCM prediction is used (active) for each encoded subband of each primary audio channel. It is transmitted even for VQ encoded subbands.

PVQ

PVQ indexes to the vector codebook, (i.e. the same code book as in the core), to get the ADPCM prediction coefficients. It is transmitted only for subbands whose ADPCM is active.

ABITS96

ABITS96[ch][n] are first difference-encoded. If HIGHRESFLAG96K = 0, the ABITS96 are then Huffman encoded using the 17 level codebooks, (clause D.5.8), otherwise they use the 33 level codebooks (clause D.5.10). This encoding is the same for all subbands (including the subbands that are VQ encoded) of each channel and is conveyed by BHUFF96. The index obtained after Huffman decoding indicates the quantizer that was used to quantize the subband samples for the n^{th} subband of channel ch.

SCALES96

The quantization indices of the scale factors are encoded by 129-level Huffman codebooks (clause D.5.12). They are also difference-encoded before Huffman coding. The scale factors are obtained by using the quantization indexes to look up the 6-bit square-root quantization table (see Table 5-24). Single scale factor is transmitted per each active extension subband. The scale factors are transmitted even for the subbands with ABIT=0.

JOIN_SHUFF96

If joint subband coding is enabled, (JOINX96[ch]>0),then JOIN_SHUFF96[ch] selects which code book was used to encode JOIN_SCALES96 which will be used when copying subband samples from the source channel to the current channel ch. The extension joint scale factors are encoded in the same way as the extension SCALES and the codebook is obtained by Table 6-6.

JOIN_SCALES96

The scale factors are used to scale the subband samples copied from the source channel (JOINX96[ch]-1) to the current channel. The joint subbands start from the anSBE96 of the current channel until the anSBE96 of the source channel. Prior to its quantization the joint subband scale factors are normalized by the source channel scale factors in the corresponding subbands. The quantization index of the joint scale factor is encoded using the Huffman codebook indexed by JOIN_SHUFF96[ch]. The scale factors are obtained by using the quantization indexes to look up the 6-bit square root quantization in Table 5-24.

SICRC96

If CPF = 1 then SICRC96 shall be extracted from the bitstream. The CRC value test shall not be applied.

6.2.4.6 96 kHz Extension Audio Data Arrays

Table 6-10: Extension Audio Data Arrays

X96 Extension Audio Data Arrays	Size (Bits)

HFREQ96
for (ch=0; ch<nPCHS; ch++) {
nNumSamplSub-subFr = 8;
nSsfIter = nSSC/2;
if ((nNumSamplSub-subFr*nSSC-nSsfIter*16) !=0)
nSsfIter++;
 	
for (n=nSBS96; n<anSBE96[ch]; n++) {
rScale = real(SCALES96[ch][n][0]);
 	
switch (ABITS96[ch][n]) {
case 0:
// No bits allocated
// Generate uniformly distributed random samples in range
// [-0.5 , 0.5] and scale them with the extracted scale factor rScale
	10 bits per applicable subband

X96 Extension Audio Data Arrays	Size (Bits)

aPrmCh[ch].aSubband[n].
GenRandomSamples(nNumSamplSub-subFr*nSSC, rScale);
break;
case 1:
prSample=aPrmCh[ch].aSubband[n].raSample[NumADPCMCoeff];
for (m=0; m<nSsfIter; m++) {
// Unpack
nVQIndex = ExtractBits(10);
nNumElementVQ = nSSC*nNumSamplSub-subFr - m*16;
nNumElementVQ=
(nNumElementVQ>16) ? 16 : nNumElementVQ;
// Look up
HFreqVQ.LookUp(nVQIndex, prSample, nNumElementVQ);
// Scale up
for (Ssfiter=0; Ssfiter<nNumElementVQ; Ssfiter++)
*(prSample++) *= rScale;
}
break;
default:
} } }	

AUDIO DATA	

// Sub-sub-frame Loop
for (nSub-sub-frame=0; nSub-sub-frame<nSSC; nSub-sub-frame++) {
// Channel Loop
for (ch=0; ch<nPCHS; ch++) {
// Subband Loop
for (n=nSBS96; n<anSBE96[ch]; n++) {
nSample = nSub-sub-frame*nNumSamplSub-subFr;
nABITS = ABITS96[ch][n]-1;
 	
switch (nABITS){
case -1:
// No bits allocated
nQType = 0;
break;
case 0:
// VQ in current subband
nQType = 0;
break;
default:
// Quantizer select
nSEL = SEL96[ch][nABITS-1]; // Number of quantizers
nNumQ = pCQGroupAUDIO[nABITS-1].nNumQ-1; // Determine quantizer type
nQType = 1; // Assume Huffman quantizers as default
if (nSEL==nNumQ) {
if (nABITS<=7)
nQType = 3; // Block quantizers
else
nQType = 2; // Linear quantizer
}
pCQGroup = &pCQGroupAUDIO[nABITS-1]; // Select quantizer group
}
// Extract bits
switch (nQType) {
case 0 :
// Case of VQ or ABIT=0
break;
case 1 : // Huffman quantizers
for (m=0; m<nNumSamplSub-subFr; m++, nSample++)
pCQGroup->ppQ[nSEL]-> InverseQ(InputFrame,AUDIO[ch][n][nSample]);
break;
case 2 : // Linear quantizers
for (m=0; m<nNumSamplSub-subFr; m++, nSample++) {
pCQGroup->ppQ[nSEL]->InverseQ(InputFrame, nCode);
AUDIO[ch][n][nSample] = pCQGroup->ppQ[nSEL]->SignExtension(nCode);
}
break;
case 3 : // Block quantizers
int nResidue;
CBlockQ *pCBQ;
pCBQ = &pCBlockQ[nABITS-1]; // Select block quantizer
for (m=0; m<nNumSamplSub-subFr/4; m++) {
// Get block code
pCQGroup->ppQ[nSEL]->InverseQ(InputFrame, nCode);
// Lookup 4 samples for a single block code
nResidue=pCBQ->LookUp(nCode,&AUDIO[ch][n][nSample]);
	VARIABLE BITS

X96 Extension Audio Data Arrays	Size (Bits)

<pre>nSample += 4;
}
break;
default: // No bits allocated
} // End of Switch
} // End of subband Loop
} // End of channel Loop
 	
// Check for DSYNC
if ((nSub-sub-frame==(nSSC-1)) || (ASPF==1)) {
SYNC = InputFrame.ExtractBits(16);
if (DSYNC != 0xffff) {
nErrorFlag = 5; // 5 = sync error
printf("Wrong DSYNC %x detected at end of sub-frame %d sub-sub-frame %d\n\n",
DSYNC, nSub-frame, nSub-sub-frame);
}
}
} // End of sub-sub-frame loop
 	
N = 16*nSSC;
// Select step size table
if (RATE == 0x1f)
pStepSizeTable = &StepSizeLossLess;
else
pStepSizeTable = &StepSizeLossy;
 	
//
// Scale factor and step size
for (ch=0; ch<nPCHS; ch++) { // Channels
for (n=nSBS96; n<anSBE96[ch]; n++) { // Subbands
pSubband = &aPrmCh[ch].aSubband[n];
// Reset assembled sample index
nAssembledSampleIndex = NumADPCMCoeff;
// Bit allocation
nABITS = ABITS96[ch][n]-1;
 	
if (nABITS>0) {
// Look up step size
pStepSizeTable->LookUp(nABITS, rStepSize);
// Scale factor
rStepRMS = rStepSize * (real)SCALES96[ch][n][0];
for (m=0; m<N; m++, nAssembledSampleIndex++)
pSubband->raSample[nAssembledSampleIndex]= rStepRMS*AUDIO[ch][n][m];
}
// Inverse ADPCM
if (PMODE[ch][n] != 0)// Only when prediction mode is on.
pSubband->InverseADPCM(2*CSubband::nNumSample);
 	
} // End of subband loop
} // End of channel Loop
// Update ADPCM history
for (ch=0; ch<nPCHS; ch++)
aPrmCh[ch].UpdateADPCMHistory();
 	
// Joint intensity coding and clear unused subbands
for (ch=0; ch<nPCHS; ch++) { // Channels
if (JOINX96[ch]>0) { // Joint subbands
// Copy joint subbands
nSourceCh = JOINX96[ch]-1;
for (n=anSBE96[ch]; n<anSBE96[nSourceCh]; n++) {
rJScale = JOIN_SCALES96[ch][n];
pSSubband= &(aPrmCh[nSourceCh].aSubband[n]); // Source subband
pSubband = &(aPrmCh[ch].aSubband[n]); // Joint subband
for (m=0; m<NumADPCMCoeff+N; m++)
pSubband->raSample[m] = rJScale*pSSubband->raSample[m];
}
// Clear unused subbands
for (n=anSBE96[nSourceCh]; n<NumSubband; n++) { // Subbands
pSubband = &(aPrmCh[ch].aSubband[n]);
for (m=0; m<NumADPCMCoeff+N; m++)
pSubband->raSample[m] = (real)0;
}
}
else { // No joint subbands
// Clear unused subbands
for (n=anSBE96[ch]; n<NumSubband; n++) { // Subbands
pSubband = &(aPrmCh[ch].aSubband[n]);</pre>
	10 bits per applicable subband

X96 Extension Audio Data Arrays	Size (Bits)

for (m=0; m<NumADPCMCoeff+N; m++)
pSubband->raSample[m] = (real)0;
}
}
} // End channel loop
//
// Clear unused subbands
for (ch=0; ch<nPCHS; ch++) { // Channels
for (n=0; n<nSBS96; n++) { // Subbands
pSubband = &(aPrmCh[ch].aSubband[n]);
for (m=0; m<NumADPCMCoeff+N; m++)
pSubband->raSample[m] = (real)0;
}
}	

HFREQ96

Some high frequency extension subbands are encoded using vector quantization (VQ). The encoder searches for the 32-element vector with elements 0 to 15 that best matches the vector of 16 subband samples, corresponding to the 16 samples (at 96 kHz/64) from the current subsubframe. This vector is indexed by HFREQ96. One HFREQ06 is transmitted per one VQ encoded extension subband in each subsubframe. The 10-bit index HFREQ96 points to one of 1 024 vectors each consisting of 32 elements (the VQ table used here is the same as the one used in the core HFREQ VQ see Page 78).

NOTE: In the subbands with no allocated bits (ABIT=0) the subband samples are generated from the random samples and this operation is included in the pseudo code in Table 6-10.

AUDIO

The audio data are grouped as nSSC subsubframes, each consisting of 16 samples for each subband. A mid-tread linear quantizer indexed by ABITS quantizes each sample. The resultant quantization index may further be encoded by either a Huffman or block code. If it is not, it is included in the bitstream as 2's compliment. All this information is indicated by SEL. The (ABITS, SEL) pair then tells how the subband samples should be extracted from the bitstream.

The resultant subband samples are then compensated by their respective quantization step sizes and scale factors. For the subbands that are ADPCM encoded, the samples of these subbands thus far obtained are actually the difference signals. Their real values shall be recovered through a reverse ADPCM process. In the subbands with ABIT=0, no audio codes are transmitted.

At the end of each subsubframe, there may be a synchronization check word DSYNC = 0xffff depending on the flag ASPF in the frame header, but there shall be at least a DSYNC at the end of each subframe.

6.2.4.7 Interpolation of the LFE Channel Samples

On the DTS encoder side, the LFE channel samples are encoded in the core encoder. No effort is made to represent higher frequency components or higher resolution in the extension encoder. Encoder first decimates input LFE samples by a factor of 64x resulting in an effective channel bandwidth of 375 Hz (352 Hz in case of 44,1 kHz sampling). These samples are furthermore quantized and transmitted in a DTS core bit-stream. Core decoder performs inverse quantization and 64x interpolation that generates reconstructed LFE samples at 48 kHz (44,1 kHz). The reconstructed LFE samples have significant frequency components of up to 375 Hz (352 Hz) and all other frequency components are at least 100 dB lower in levels. The LFE samples at 96 kHz (88,2 kHz) are generated in an extension decoder using interpolation by factor of 2x. The goal of this filter is to remove an image introduced by interpolation. Significant frequency components of this image lie between 47,625 kHz (43,748 kHz) and 48 kHz (44,1 kHz). Thus, a simple 5-tap linear phase FIR filter with a magnitude response shown in Figure 6-6 is used as the 2x interpolation filter in LFE channel.

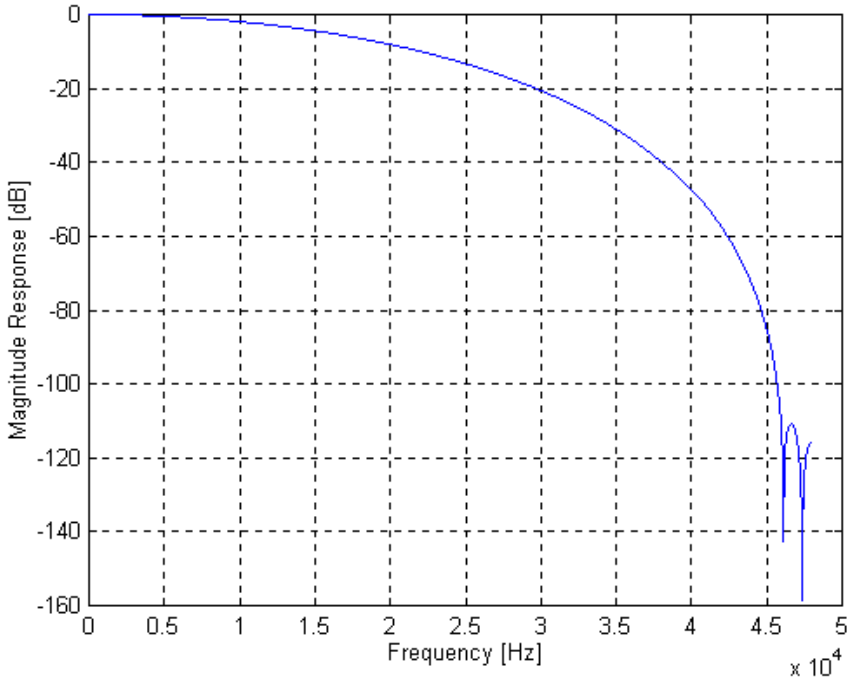


Figure 6-6: Magnitude Response of LFE Interpolation 2x Filter at 96 kHz

The coefficients of 2x LFE interpolation filter scaled by an interpolation factor of 2 are given in Table 6-11.

Table 6-11: LFE 2x Interpolation Filter Coefficients

1,2553677676342990e-001
4,9999913800216800e-001
7,4892817046880420e-001
4,9999913800216800e-001
1,2553677676342990e-001

The near-perfect reconstruction 64-band cosine modulated filter bank is obtained by modulating the 1 024-tap FIR linear phase prototype filter. The signs of these filter coefficients were changed in a manner appropriate for an efficient implementation of polyphase filtering. In particular, the signs of all coefficients in every even indexed block of 128 coefficients are changed, e.g. coefficients in ranges 129 to 256, 385 to 512, 641 to 768 and 897 to 1 024 changed their signs. The modified prototype filter coefficients are given in clause D.9.

6.3 XBR - Extended Bit Rate Extension

6.3.1 About the XBR Extension

The original DTS core encoder allowed 5.1 channels of high quality audio (sampling frequency F_s up to 48 kHz) to be encoded and subsequent 'extensions' to the core were added to allow (a) additional channels to be encoded and (b) high frequency elements to be encoded (Fs up to 96 kHz).

This clause describes a Bit-rate Extension (XBR) that allows for the total core + extensions bit-rates to be larger than 1,5 Mbps. This in itself is nothing new, but the difference is a *guaranteed backward compatibility with the DTS core decoders already in the market* (which can only handle DTS bit streams up to rates of 1,5 Mbps). This is achieved by the inclusion of the core substream at 1,5 Mbps, along with the extension substream that carries the XBR extension. The XBR extension enhances the quality of audio that has been encoded in the core substream, by means of allocating additional bit-pool for the encoding of residual signals. The residual signals carry the information about the original audio that has not been represented by the core substream data.

6.3.2 DTS Core Substream Encoder + XBR Extension Encoder

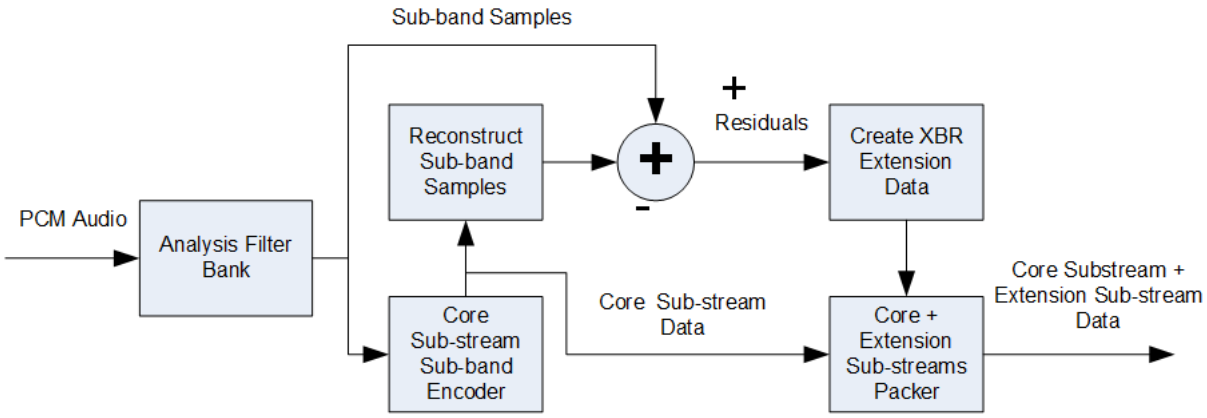


Figure 6-7: DTS Core Substream + XBR Extension Encoder

The XBR extension involves encoding the *residuals* that are the difference between the original subband samples and the subband samples reconstructed by decoding the core substream data.

The XBR encoding only occurs after the core substream encoding has completed. Therefore the XBR encoding has NO effect on the core substream encoding.

The XBR encoding process is as follows:

- **Generate residuals.** These values are just the difference between the original subband samples and the subband samples reconstructed by decoding the core substream data.
- **Bit allocation.** Bits are then allocated in the bit rate extension. The subband power of the residuals is calculated and based on the XBR bit-pool. It divides it to nNumSbFrm by nXBRChannels, bit-pools one per subframe, in each channel. Division is uniform between the subframes and proportional to the channel power. The residual samples overwrite the core subband samples in psub_band_samples.
- **Quantize scales.** The scale factors are then quantized using Table 5-24.
- **Quantize samples.** The (residual) subband samples are then quantized.
- **Pack XBR extension.** The encoded XBR data is then packed into the DTS frame.

6.3.3 DTS XBR Bit Rate Extension Decoder

In the decoder, the unpack block first separates the core substream and the extension subs-stream. Next the core substream data is decoded and the corresponding subband samples are reconstructed. The XBR decoder then assembles the residual subband samples carried in the XBR extension and adds them to the corresponding subband samples that have been reconstructed from the core substream data, as illustrated in Figure 6-8. The resulting subband samples are fed to the synthesis filter bank, where the decoded PCM audio samples are synthesized. For legacy decoders, which are not capable of decoding the XBR extension, the XBR data is ignored and the subband samples that have been reconstructed from the core substream data are fed to the synthesis filter bank, where the core substream decoded PCM audio samples are synthesized.

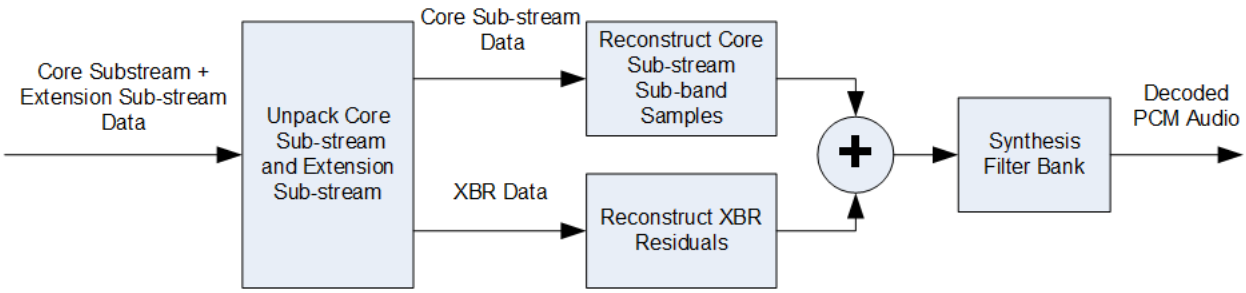


Figure 6-8: DTS XBR decoder preliminary unpacking

Code operation:

UnpackHeaderXBR - extract XBR header information including TMODE bit. Set up pointers for the audio samples that follow.

UnpackXBRSub-frame - unpack ABITS information and generate scale factors. Extract audio samples and scale them.

AssembleXBRSubbands - assemble together residual subband samples adding them to the corresponding subband samples reconstructed from the core substream data.

6.3.4 Extension (XBR) Bitstream Components

The frame of XBR data is divided into a frame header and up to four channel sets. Each channel set has its own channel set header, which are all grouped together at the end of the frame header as shown in Figure 6-9. The CRC word is included at the end of frame header to allow detection of errors in the frame header.

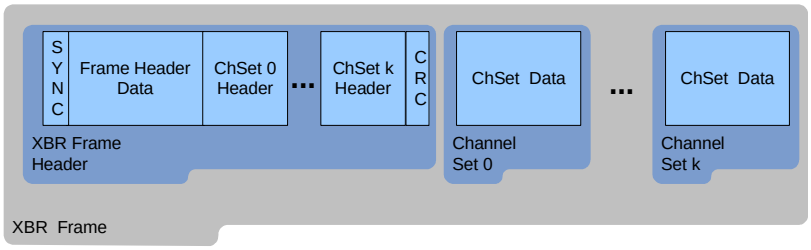


Figure 6-9: XBR Data Frame Structure

The channel set data is further subdivided into the subframes. Each subframe consists of a subframe header and the audio data. The audio data is organized in subsubframes as shown in Figure 6-10. The number of subframes is the same for all channel sets and is equal to the number of subframes in the core frame. Similarly the number of subsubframes is the same for all subframes of all channel sets and is equal to the number of subsubframes within each subframe of the core frame. In other words, the subframe and subsubframe partitioning within the XBR frame follows the partitioning present in the core frame.

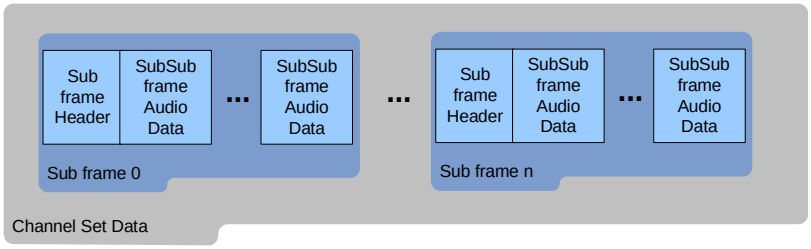


Figure 6-10: Channel Set Data Structure

6.3.5 XBR Frame Header

Table 6-12: XBR Frame Header Structure

XBR Frame Header Syntax	Size (Bits)
`SYNCXBR = ExtractBits(32);`	32
`nHeaderSizeXBR = ExtractBits(6)+1;`	6
`nuNumChSetsInXBR = ExtractBits(2)+1;`	2
`for (nChSet=0; nChSet < nuNumChSetsInXBR; nChSet ++)` `  pnuChSetFsize[nChSet] = ExtractBits(14)+1;`	14
`nXBRTMODEFlag = (ExtractBits(1) == 1) ? true : false;`	1
`for (nChSet=0; nChSet < nuNumChSetsInXBR; nChSet ++)` `  Extract ChannelSetSubHeaderXBR{}`	See XBR Channel Set Sub-Header Table
`ReservedHeaderXBR = ExtractBits(…);`	…
`ByteAlignHeaderXBR = ExtractBits(0 … 7);`	0...7
`nCRC16HeaderXBR = ExtractBits(16);`	16

SYNCXBR (XBR Sync Word)

The DWORD aligned XBR synchronization word has value 0x655e315e. During sync detection the nCRC16HeaderXBR checksum is used to further verify that the detected sync pattern is not a random alias.

nHeaderSizeXBR (XBR frame header length)

The size of the header in bytes from the SYNCXBR to the nCRC16HeaderXBR inclusive. This value determines the location of the channel set data block in the first channel set. This marker also designates the end of the field nCRC16HeaderXBR and allows quick location of the checksum at byte position nHeaderSizeXBR-2.

nuNumChSetsInXBR (Number of Channel Sets)

All channels within the XBR extension are organized in individually decodable channel sets. The nNumChSetaInXBR is the number of channel sets that are present in XBR.

pnuChSetFsize (Channel Set Data Byte Size)

The pnuChSetFsize[nChSet] indicates the total number of bytes in the data portion of the channel set (nChSet) in the XBR frame. Starting from the SYNCXBR and using the cumulative sum of nHeaderSizeXBR and the pnuChSetFsize[k] (over all channel sets k=0, … nChSet -1) as offset, decoder may traverse to the beginning of channel set data block in the channel set nChSet.

XBRTMODEFlag (TMODE used flag)

If the XBRTMODEFlag is true, the TMODES that are present in the core stream (and if present in the XCH stream or XXCH stream) shall be used for extraction of the XBR stream.

If XBRTMODEFlag is false, the TMODES present in the core channel extension stream shall be ignored by the XBR decoder.

ReservedHeaderXBR (Reserved)

This field is reserved for additional XBR header information. The decoder shall assume that this field is present and of unspecified length. Therefore, in order to continue unpacking the stream, the decoder shall skip over this field using the XBR header start pointer and the XBR header size nHeaderSizeXBR.

ByteAlignHeaderXBR (Pad to BYTE boundary)

This field ensures that the CRC16 field that follows is aligned to a byte boundary to allow fast table based CRC16 calculation. Append '0's until bit position is a multiple of 8.

nCRC16HeaderXBR (CRC16 of XBR frame header)

It represents the 16-bit CRC check word of the entire XBR header from positions nHeaderSizeXBR to ByteAlignHeaderXBR inclusive.

6.3.6 XBR Channel Set Sub-Header

Table 6-13: XBR Channel Set Sub-Header Structure

XBR Channel Set Sub-Header Syntax	Size (Bits)
nXBRCh = ExtractBits(3) + 1;	3
nBits4MaxSubbands = ExtractBits(2) + 5;	2
for (nCh=0; nCh< nXBRCh ; nCh++) naXBRActiveBands = ExtractBits(nBits4MaxSubbands) + 1;	nBits4MaxSubbands

nXBRCh (Number of XBR channels)

nXBRCh indicates the number of channels that are encoded in this XBR channel set.

nBits4MaxSubbands (Max subbands flag)

The nBits4MaxSubbands flag indicates the number of bits used to indicate the number of active subbands in the XBR channel set. From this, the number of bits used for indicating the number of active subbands in each channel (naXBRActiveBands[ch]) can be calculated. This is a 2-bit field with [0,1,2,3] corresponding to values [5,6,7,8].

naXBRActiveBands (XBR Active subbands)

This field indicates the number of subbands that are encoded in a particular channel of the XBR channel set. Encoding always starts from subband 0 and it goes up to the subband XBR_ActiveBands-1 (XBR_ActiveBands $\leq$ 32 for Fs $\leq$ 48 k or XBR_ActiveBands $\leq$ 64 for 48 kHz < Fs $\leq$ 96 kHz).

6.3.7 XBR Channel Set Data

6.3.7.1 XBR Channel Set Syntax

Table 6-14: XBR Channel Set Data Syntax

XBR Channel Set Data Syntax	Size (Bits)

for (nCh=0; nCh<nXBRCh; nCh++) {
nNumABITSbits [nCh] = ExtractBits(2)+2;
}
	2 bits per channel

// Unpack ABITSH
nChSBIndex = 0;
for (nCh=0; nCh<nXBRCh; nCh++){
nTmp = nNumABITSbits [nCh];
for (nSB=0; nSB<naXBRActiveBands[nCh]; nSB++, nChSBIndex++){
ancAbitsHigh[nChSBIndex] = ExtractBits(nTmp);
}
} // end of channel loop
	Variable bits per channel

for (nCh=0; nCh<nXBRCh; nCh++){
ancTemp[nCh] = InputFrame.ExtractBits(3);
if (ancTemp[nCh]<1)
return SYNC_ERROR_DEF;
}
	3 bits per channel

// Generate scale factors
nChSBIndex = 0;
for (nCh=0; nCh<nXBRCh; nCh++){
// Select RMS table
if (SHUFF[nCh] == 6)
pScaleTable = &RMS7Bit;
else
pScaleTable = &RMS6Bit;
// nTmp is the number of bits used for each scale index
nTmp = ancTemp[nCh];
 	
for (nSB=0; nSB<naXBRActiveBands[nCh]; nSB++, nChSBIndex++){
if (ancAbitsHigh[nChSBIndex]>0){
// Pre-transient scale
// Unpack Scale index
nScaleInd = ExtractBits(nTmp);
 	
// Look-up scale factor
if (pScaleTable->LookUp(nScaleInd, anScalesHigh[nChSBIndex<<1])==NULL
) {
	Variable per subband

XBR Channel Set Data Syntax	Size (Bits)

`return SYNC_ERROR_DEF;`
`}`
`// Post-transient scale`
`if (nXBRTMODEFlag*TMODE[nCh][nSB]>0){`
`// Unpack Scale index`
`nScaleInd = ExtractBits(nTmp);`
 	
`// Look-up scale factor`
`if ( pScaleTable->LookUp(nScaleInd,`
`anScalesHigh[(nChSBIndex<<1)+1])==NULL ) {`
`return SYNC_ERROR_DEF;`
`}`
`} // if (ancAbitsHigh[nChSBIndex]>0)`
`else{`
`anScalesHigh[nChSBIndex<<1] = 0;`
`anScalesHigh[(nChSBIndex<<1)+1] = 0;`
`}`
`}`
`}`	

6.3.7.2 Subframe Side Information

The pseudo code for the subframe side information is depicted in Table 6-14.

Number of bits for ABITS nNumABITSbits

Indicates the number of bits used for encoding ABITSH. This is a 2-bit field with [0,1,2,3] corresponding to values [2,3,4,5].

XBR Bit allocation Index (ABITSH)

ABITSH indicates the bit allocation indexes for all channels in the channel set and all subbands in the Bit-rate extension.

Number of bits for Scale indices (nNumScalesbits)

nNumScalesbits indicates the number of bits used for encoding Scale indices (3-bit field). The encoder uses either 6-bit or 7-bit square-root quantization table (as indicated by SHUFF[nCh]) to quantize the scale factors. The quantization indexes (6-bit or 7-bit) values are further analysed to determine the largest index value (Max_Scale_Ind) for each channel in the current subframe. The efficient transmission of scale indexes is achieved by using nNumScalesbits = ceil($\log_2$(Max_Scale_Ind)) bits per index.

XBR Scales indices (nScaleInd)

First the scale factor quantization indexes are extracted as the nNumScalesbits bit words. The scale factors are obtained by using the quantization indexes to look up the square-root quantization table (this is the same table as for the core scale factors). Two scale factors are transmitted per each active extension subband if TMODE and nXBRTMODEFlag are set. The scale factors are not transmitted for the subbands with ABITSH=0.

6.3.7.3 XBR Extension Residual Audio Data Arrays

Table 6-15: XBR Extension Residual Audio Data

XBR Extension Residual Audio Data	Size (Bits)
AUDIO	

`// Unpack residual audio codes`
`for (nSub-sub-frame=0; nSub-sub-frame<nSSC; nSub-sub-frame++) {`
`nChSBIndex = 0;`
`for (nCh=0; nCh<nXBRCh; nCh++){ // Channels`
`// Subbands`
`for (nSB=0; nSB<naXBRActiveBands[nCh]; nSB++, nChSBIndex++) {`
`nXBR_ABITS = ancAbitsHigh[nChSBIndex];`
`nSmpl = nSub-sub-frame<<3;`
`if (nXBR_ABITS>0){ // Audio codes are present only for nIndex>0`
`if (nXBR_ABITS>7){`
`// Linear quantizers`
`nTmp = nXBR_ABITS-3; // Number of bits used for packing`
	Variable per subband

XBR Extension Residual Audio Data	Size (Bits)

nShift = 32 - nTmp;
for (nn=0; nn<8; nn++, nSmpl++) {
// Extract audio residuals and do the sign extension
nCode = InputFrame.ExtractBits(nTmp)<<nShift;
AUDIO[nCh][nSB][nSmpl] = nCode>>nShift;
}
} // end of nXBR_ABITS>7
else{
// Block encoded quantizers
pCBQ = &pCBlockQ[nXBR_ABITS-1]; // Select block quantizer
// Get the length of one block code
nTmp = nLenBlockCode[nXBR_ABITS-1];
for (nn=0; nn<2; nn++) {
nCode = InputFrame.ExtractBits(nTmp);
nResidue = pCBQ->LookUp(nCode,&AUDIO[nCh][nSB][nSmpl]);
// Lookup 4 samples for a single block code
if (nResidue != 0) {
printf("XBR AUDIO: Block code error\n");
return SYNC_ERROR_DEF;
}
nSmpl += 4;
}
} // end of nXBR_ABITS<=7
} // end of nXBR_ABITS>0
else{ // nXBR_ABITS=0
for (nn=0; nn<8; nn++, nSmpl++)
AUDIO[nCh][nSB][nSmpl] = 0;
} // end of nIndex condition
} // End nSB loop
} // End nCh loop
// Check for DSYNC which is used to verify end of sub-frame or sub-sub-frame position
if ((nSub-sub-frame==(nSSC-1)) || (ASPF==1)) {
DSYNC = InputFrame.ExtractBits(16);
if (DSYNC != 0xffff) {
return SYNC_ERROR_DEF;
}
}
} // end nSub-sub-frame loop	

AUDIO (Audio Data)

The residual audio samples are grouped as nSSC subsubframes, each consisting of eight samples for each subband. All samples in each subband of each channel may have been linear quantized, or block encoded - this is indicated by nXBR_ABITS. If nXBR_ABITS = 0 then no audio residuals are present in the DTS stream for this subsubframe of this channel and subband. The corresponding residual array entries should be filled with zeros.

At the end of each subsubframe there may be a synchronization check word DSYNC = 0xffff depending on the flag ASPF in the frame header, but there shall be at least a DSYNC at the end of each subframe.

6.3.8 Assembly of XBR subbands

As per Figure 6-8, once the XBR information has been extracted, it shall be recombined with the 1,5 Mbps backward compatible 'core + extensions' stream prior to decoding of these streams. Table 6-16 shows the pseudo code implementation.

Table 6-16: XBR Assembling Subbands

XBR Assembling Subbands

//Assembling subbands following XBR extraction and prior to core + extensions decoding
// This function assembles together residual subband samples
// and adds them to the corresponding core+XCh+X96 subband samples
// stored in CSB.naSample[] arrays
AssembleXBRSubbands(unsigned int nBands2Use)
{
DTS__int64 dAcc;
 	
// Select step size table
pStepSizeTable = &StepSizeLossLess;

XBR Assembling Subbands

```
// Assemble subbands
nAccChSBInd = 0;
for (nCh=0; nCh<nXBRCh; nCh++) {              // Channels
  nBands2Use = (nBands2Use>naXBRActiveBands[nCh]) ? naXBRActiveBands[nCh] : nBands2Use;
  nChSBIndex = nAccChSBInd;
  for (nSB=0; nSB<nBands2Use; nSB++, nChSBIndex++) {          // Subbands
    switch (nX96Present){
    case 0:
pSubbandSmpls = aPrmCh[nCh].aSubband[nSB].naSampleCore;
break;
    default:
pSubbandSmpls = aPrmCh[nCh].aSubband[nSB].naSampleExt;
break;
    }
    // Look up step size
    nABITS = ancAbitsHigh[nChSBIndex];
    if ( pStepSizeTable->LookUp(nABITS, nStepSize)==NULL )
printf("ERROR: StepSize lookup failure --- ABITS=%d", nABITS);

    // Set transient (sub-sub-frame) location
    nTmode = nXBRTMODEFlag*TMODE[nCh][nSB];

    nSmplLim = ( nTmode == 0 ) ? CSubband::nNumSample : 8*nTmode;

    // Reconstruct residuals and add them to the existing (in naSample array)
    // subband samples; Store results back to the naSample array

    nRawSampleIndex=0;                        // Reset raw sample index
    nAssembledSampleIndex = NumADPCMCoeff;    // Reset assembled sample index
    // PreTransient
    dAcc = (DTS__int64) nStepSize * anScalesHigh[nChSBIndex<<1];

    // Find location of the most significant "1"
    m=0;
    while (dAcc>0){
m++;
dAcc=dAcc>>1;
    }
    nShift = (m>SUBBSAMPLS_QRES) ? (m-SUBBSAMPLS_QRES) : 0;
    nStepRMS = (int) ( ( ((DTS__int64) nStepSize * anScalesHigh[nChSBIndex<<1]))>>nShift );
    nShift = 22-nShift;
    nRoundF = (nShift>0) ? 1<<(nShift-1) : 0;

    for (m=0; m<nSmplLim; m++, nRawSampleIndex++, nAssembledSampleIndex++){
dAcc = nRoundF + (DTS__int64) nStepRMS*AUDIO[nCh][nSB][nRawSampleIndex];
//limits check
nTmp = (int) (dAcc>>nShift);
if (nTmp>nHigh)
  nTmp = nHigh;
else if (nTmp<nLow)
  nTmp = nLow;

//write assembled value
pSubbandSmpls[nAssembledSampleIndex] += nTmp;
    }
    // AfterTransient complete the same process as for pretransient
  } // SB loop
  nAccChSBInd += naXBRActiveBands[nCh];
}   // channel loop
return 1;
```

6.4 Extension to 6.1 Channels (XCh)

6.4.1 About the XCh Extension

The XCh extension expands the core capability to encoding of 6.1 discrete channels in the configuration that assumes standard 5.1 layout plus the additional surround channel positioned directly behind the listener (180°) and denoted as a centre surround (Cs) channel. Additional requirement for the valid XCh stream is that 6.1 to 5.1 down-mix is embedded in the stream, i.e. the Cs channel is mixed into the left surround (Ls) and right surround (Rs) channels with the attenuation of -3 dB exclusively. This 5.1 down-mix is encoded in the core stream and the Cs channel is encoded in the XCh stream. This way in the 5.1 listening environment the Cs channel will be reproduced as the phantom image between the Ls and Rs speakers. The Cs channel in the XCh stream is compressed using exactly the same technology as the core audio channels. The audio data representing this extension channel (XCh stream) is appended to the end of the core audio data (core stream). This extension audio data is automatically ignored by first generation DTS decoders but can be decoded by second generation DTS decoders.

6.4.2 Unpack Frame Header

Table 6-17: XCH Frame header

XCH Frame Header	Size (Bits)
XChSYNC = ExtractBits(32);	32
XChFSIZE = ExtractBits(10);	10
AMODE = ExtractBits(4);	4

XChSYNC (Channel Extension Sync Word)

The synchronization word XChSYNC = 0x5a5a5a5a for the channel extension audio comes after all other extension streams (i.e. in case of multiple extension streams the XCh stream is always the last). For 16-bit streams, XChSYNC is aligned to a 32-bit word boundary. For 14-bit streams, it is aligned to both 32-bit and 28-bit word boundaries, meaning that the sync word appears as 0x1696e5a5 in the 28-bit stream and as 0x5a5a5a5a after this stream is packed into a 32-bit stream.

Since the pseudo sync word might appear in the bitstream, it is MANDATORY to check the distance between this sync and the end of the encoded bitstream. This distance in bytes should be equal to XChFSIZE+1. The parameter XChFSIZE is described below.

NOTE: For compatibility reasons with legacy bitstreams the estimated distance in bytes is checked against the XChFSIZE+1 as well as the XChFSIZE. The XCh synchronization is pronounced if the distance matches either of these two values.

XChFSIZE (Primary Frame Byte Size)

(XChFSIZE+1) is the distance in bytes from current extension sync word to the end of the current audio frame. Valid range for XChFSIZE: 95 to 1 023. Invalid range: 0 to 94.

AMODE (Extension Channel Arrangement)

Audio channel arrangement which describes the number of audio channels (CHS) and the audio playback arrangement. It is set to represent the number of extension channels.

6.4.3 Unpack Audio Header

Table 6-18: XCH Audio header

XCH Audio Header	Size (Bits)

PCHS = ExtractBits(3);
nPCHS = PCHS + 1;
for (ch=0; ch<nPCHS; ch++) {
  SUBS[ch] = ExtractBits(5);
  nSUBS[ch] = SUBS[ch] + 2;
}
	3
	
	5 bits per channel

XCH Audio Header	Size (Bits)

for (ch=0; ch<nPCHS; ch++) {
  VQSUB[ch] = ExtractBits(5);
  nVQSUB[ch] = VQSUB[ch] + 1;
}
	5 bits per channel

for (ch=0; ch<nPCHS; ch++) JOINX[ch] = ExtractBits(3);	3 bits per channel
for (ch=0; ch<nPCHS; ch++) THUFF[ch] = ExtractBits(2);	2 bits per channel
for (ch=0; ch<nPCHS; ch++) SHUFF[ch] = ExtractBits(3);	3 bits per channel
for (ch=0; ch<nPCHS; ch++) BHUFF[ch] = ExtractBits(3);	3 bits per channel

// ABITS=1:
n=0;
for (ch=0; ch<nPCHS; ch++)
  SEL[ch][n] = ExtractBits(1);
// ABITS = 2 to 5:
for (n=1; n<5; n++)
  for (ch=0; ch<nPCHS; ch++)
  SEL[ch][n] = ExtractBits(2);
// ABITS = 6 to 10:
for (n=5; n<10; n++)
  for (ch=0; ch<nPCHS; ch++)
  SEL[ch][n] = ExtractBits(3);
// ABITS = 11 to 26:
for (n=10; n<26; n++)
  for (ch=0; ch<nPCHS; ch++)
  SEL[ch][n] = 0; // Not transmitted, set to zero.
// ABITS = 1 :
n = 0;
for (ch=0; ch<nPCHS; ch++)
  if (SEL[ch][n] == 0) { // Transmitted only if SEL=0 (Huffman code used)
    // Extract ADJ index
    ADJ = ExtractBits(2);
    // Look up ADJ table
    arADJ[ch][n] = AdjTable[ADJ];
  }
// ABITS = 2 to 5:
for (n=1; n<5; n++)
  for (ch=0; ch<nPCHS; ch++)
    if (SEL[ch][n] < 3) { // Transmitted only when SEL<3
      // Extract ADJ index
      ADJ = ExtractBits(2);
      // Look up ADJ table
      arADJ[ch][n] = AdjTable[ADJ];
    }
// ABITS = 6 to 10:
for (n=5; n<10; n++)
  for (ch=0; ch<nPCHS; ch++)
    if (SEL[ch][n] < 7) { // Transmitted only when SEL<7
      // Extract ADJ index
      ADJ = ExtractBits(2);
      // Look up ADJ table
      arADJ[ch][n] = AdjTable[ADJ];
    }
	variable bits
	2 bits per occasion

if (CPF==1) // Present only if CPF=1.
  AHCRC = ExtractBits(16);
	16

PCHS (Number of Extension Channels)

This field indicates that there are nPCHS = PCHS+1 < 5 extension audio channels in the current frame. If AMODE flag indicates more than 5 channels apart from LFE, the additional channels are the extended channels and are packed separately in the extended data arrays.

SUBS (Subband Activity Count)

This field indicates that there are nSUBS[ch] = SUBS[ch]+2 active subbands in the audio channel ch. Samples in subbands above nSUBS[ch] are zero, provided that intensity coding in that subband is disabled.

VQSUB (High Frequency VQ Start Subband)

This field indicates that high frequency samples starting from subband nVQSUB[ch]=VQSUB[ch]+1 are VQ encoded. High frequency VQ is used only for high frequency subbands, but it may go down to low frequency subbands for such audio episodes as silence. In case of insufficient MIPS, the VQs for the highest frequency subbands may be ignored without causing audible distortion.

JOINX (Joint Intensity Coding Index)

This field, JOINX[ch], indicates if joint intensity coding is enabled for channel ch and which audio channel is the source channel from which channel ch will copy subband samples. It is assumed that the source channel index is smaller than that of the intensity channel, (see Table 5-23).

THUFF (Transient Mode Code Book)

This field indicates which Huffman codebook was used to encode the transient mode data TMODE (See Table 5-23).

SHUFF (Scale Factor Code Book)

The scale factors of a channel are quantized nonlinearly using either a 6-bit (64-level, 2,2 dB per step) or a 7-bit (128-level, 1,1 dB per step) square root table, depending on the application. The quantization indexes may be further compressed by one of the five Huffman codes and this information is transmitted to the decoder by SHUFF[ch] (Table 5-24).

BHUFF (Bit Allocation Quantizer Select)

This field indicates the codebook that was used to encode the bit allocation index ABITS (to be transmitted later). See (Table 5-25).

SEL (Quantization Index Codebook Select)

After subband samples are quantized using a mid-tread linear quantizer, the quantization indexes may be further encoded using either entropy (Huffman) or block coding in order to reduce bit rate. Therefore, the subband samples may appear in the bit stream as plain quantization indexes (no further encoding), entropy (Huffman) codes, or block codes. For channel ch, the selection of a particular codebook for a mid-tread linear quantizer indexed by ABITS[ch] is transmitted to the decoder as SEL[ch][ABITS[ch]]. No SEL is transmitted for ABITS[ch]≥11, because no further encoding is used for those quantizers. The decoder can find out the particular codebook that was used using ABITS[ch] and SEL[ch][ABITS[ch]] to look up (see Table 5-26).

ADJ (Scale Factor Adjustment Index)

A scale factor adjustment index is transmitted whenever a SEL value indicates a Huffman codebook. This index points to the adjustment values shown in Table 5-27. This adjustment value should be multiplied by the scale factor (SCALE).

AHCRC (Audio Header CRC Check Word)

If CPF = 1 then AHCRC shall be extracted from the bitstream. The CRC value test shall not be applied.

6.4.4 Unpack Subframes

6.4.4.1 Side Information

Table 6-19: XCH side information

XCH audio side information	Size (Bits)

`for (ch=0; ch<nPCHS; ch++)`
`  for (n=0; n<nSUBS[ch]; n++)`
`    PMODE[ch][n] = ExtractBits(1);`
	1 bit per subband

`int nVQIndex;`
`for (ch=0; ch<nPCHS; ch++)`
`  for (n=0; n<nSUBS[ch]; n++)`
`    if ( PMODE[ch][n]>0 ) { // Transmitted only when ADPCM active`
`    // Extract the VQindex`
`      nVQIndex = ExtractBits(12);`
`      // Look up the VQ table for prediction coefficients.`
`      ADPCMCoeffVQ.LookUp(nVQIndex, PVQ[ch][n]) // 4 coefficients`
	12 bits per occurrence

XCH audio side information	Size (Bits)

}
 	
for (ch=0; ch<nPCHS; ch++) {
// BHUFF tells which codebook was used
nQSelect = BHUFF[ch];
// Use this codebook to decode the bit stream for ABITS[ch][n]
for (n=0; n<nVQSUB[ch]; n++) // Not for VQ encoded subbands.
QABITS.ppQ[nQSelect]->InverseQ(InputFrame, ABITS[ch][n])
}
// Always assume no transition unless told
for (ch=0; ch<nPCHS; ch++)
for (n=0; n<NumSubband; n++)
TMODE[ch][n] = 0;
// Decode TMODE[ch][n]
if (nSSC>1) {// Transient possible only if more than one sub-sub-frame.
for (ch=0; ch<nPCHS; ch++) {
// TMODE[ch][n] is encoded by a codebook indexed by THUFF[ch]
nQSelect = THUFF[ch];
for (n=0; n<nVQSUB[ch]; n++) // No VQ encoded subbands
if (ABITS[ch][n] >0) // Present only if bits allocated
// Use codebook nQSelect to decode TMODE from the bitstream
QTMODE.ppQ[nQSelect]>InverseQ(InputFrame,TMODE[ch][n])
}
}
for (ch=0; ch<nPCHS; ch++) {
// Clear SCALES
for (n=0; n<NumSubband; n++) {
SCALES[ch][n][0] = 0;
SCALES[ch][n][1] = 0;
}
// SHUFF indicates which codebook was used to encode SCALES
nQSelect = SHUFF[ch];
// Select the root square table (SCALES were nonlinearly
// quantized).
if (nQSelect == 6)
pScaleTable = &RMS7Bit; // 7-bit root square table
else
pScaleTable = &RMS6Bit; // 6-bit root square table
//
// Clear accumulation (if Huffman code was used, the difference
// of SCALES was encoded).
//
nScaleSum = 0;
//
// Extract SCALES for Subbands up to VQSUB[ch]
//
for (n=0; n<nVQSUB[ch]; n++)
if (ABITS[ch][n] >0) { // Not present if no bit allocated
//
// First scale factor
//
// Use the (Huffman) code indicated by nQSelect to decode
// the quantization index of SCALES from the bit stream
QSCALES.ppQ[nQSelect]->InverseQ(InputFrame, nScale);
// Take care of difference encoding
if (nQSelect < 5) // Huffman encoded, nScale is the difference
nScaleSum += nScale; // of the quantization indexes of SCALES.
else // Otherwise, nScale is the quantization
nScaleSum = nScale; // level of SCALES.
// Look up SCALES from the root square table
pScaleTable->LookUp(nScaleSum, SCALES[ch][n][0])
//
// Two scale factors transmitted if there is a transient
//
if (TMODE[ch][n]>0) {
// Use the (Huffman) code indicated by nQSelect to decode
// the quantization index of SCALES from the bit stream
QSCALES.ppQ[nQSelect]->InverseQ(InputFrame, nScale);
// Take care of difference encoding
if (nQSelect < 5) // Huffman encoded, nScale is the
nScaleSum += nScale; // of SCALES.
else // Otherwise, nScale is SCALES
nScaleSum = nScale; // itself.
// Look up SCALES from the root square table
pScaleTable->LookUp(nScaleSum, SCALES[ch][n][1])
}
}
	variable bits
	
	variable bits
	
	variable bits

XCH audio side information	Size (Bits)

//
// High frequency VQ subbands
//
for (n=nVQSUB[ch]; n<nSUBS[ch]; n++) {
// Use the code book indicated by nQSelect to decode
// the quantization index of SCALES from the bit stream
QSCALES.ppQ[nQSelect]->InverseQ(InputFrame, nScale);
// Take care of difference encoding
if (nQSelect < 5) // Huffman encoded, nScale is the
nScaleSum += nScale; // of SCALES.
else // Otherwise, nScale is SCALES
nScaleSum = nScale; // itself.
// Look up SCALES from the root square table
pScaleTable->LookUp(nScaleSum, SCALES[ch][n][0]);
}
}
for (ch=0; ch<nPCHS; ch++)
if (JOINX[ch]>0) // Transmitted only if joint subband coding enabled.
JOIN_SHUFF[ch] = ExtractBits(3);
int nSourceCh;
for (ch=0; ch<nPCHS; ch++)
if (JOINX[ch]>0) { // Only if joint subband coding enabled.
nSourceCh = JOINX[ch]-1; // Get source channel. JOINX counts
// channels as 1,2,3,4,5, so minus 1.
nQSelect = JOIN_SHUFF[ch]; // Select code book.
for (n=nSUBS[ch]; n<nSUBS[nSourceCh]; n++) {
// Use the code book indicated by nQSelect to decode
// the quantization index of JOIN_SCALES
QSCALES.ppQ[nQSelect]->InverseQ(InputFrame, nJScale);
// Bias by 64
nJScale = nJScale + 64;
// Look up JOIN_SCALES from the joint scale table
JScaleTbl.LookUp(nJScale, JOIN_SCALES[ch][n]);
}
}
if (CPF==1) // Present only if CPF=1.
SICRC = ExtractBits(16);
	3 bits per channel
	
	variable bits
	
	16

PMODE (Prediction Mode)

When PMODE[ch][n]=1, it indicates that ADPCM prediction is used (active) for subband n of extension audio channel [ch] and PMODE[ch][n]=0 otherwise. ADPCM shall be extracted from the bit stream for all subbands, but ADPCM reconstruction can be limited to the lowest 20 subbands if DSP does not have enough MIPS.

PVQ (Prediction Coefficients VQ Address)

This field indexes to the vector code book (clause D.10.1) to get the ADPCM prediction coefficients. It is transmitted only for subbands whose ADPCM is active.

ABITS (Bit Allocation Index)

This field, ABITS[ch][n], is the index to the mid-tread linear quantizer that was used to quantize the subband samples for the n^{th} subband of channel ch. ABITS[ch][n] may be transmitted as either a 4-bit or 5-bit word. When ABITS is encoded in a 4-bit word, it may be further encoded using one of the five Huffman codes. This encoding is the same for all subbands of each channel and is conveyed by BHUFF as shown in Table 5-25. There is no need to allocate bits for the high frequency subbands because they are encoded using VQ.

TMODE (Transition Mode)

This field, TMODE[ch][n], indicates if there is a transient inside a subframe (subband analysis window) for subband n of channel ch. If there is a transient (TMODE[ch][n]>0), it further indicates that the transition occurred in subsubframe (subband analysis subwindow) TMODE[ch][n] + 1. TMODE[ch][n] is encoded by one of the four Huffman codes and the selection of which is conveyed by THUFF (see Table 5-24). The decoder assumes that there is no transition (TMODE[ch][n]=0) for all subbands of all channels unless it is told otherwise by the bit stream. Transient does not occur in the following situations, so TMODE is not transmitted:

- Only one subsubframe within the current subframe. This is because the time resolution of transient analysis is a subsubframe (subband analysis subwindow).

- VQ encoded high frequency subbands. If there is a transient for a subband, it would not have been VQ encoded.
- Subbands without bit allocation. If there is no need to allocate bits for a subband, there is no need to care about transient for it.

SCALES (Scale Factors)

One scale factor is transmitted for subbands without transient. Otherwise, two are transmitted, one for the episode before the transient and the other for after the transient. The quantization indexes of the scale factors may be encoded by Huffman code as shown in Table 5-24. If this is the case, they are difference-encoded before Huffman coding. The scale factors are finally obtained by using the quantization indexes to look up either the 6-bit or 7-bit square root quantization table according to Table 5-24.

JOIN_SHUFF (Joint Subband Scale Factor Codebook)

If joint subband coding is enabled (JOINX[ch]>0), JOIN_SHUFF[ch] selects which code book was used to encode the scale factors (JOIN_SCALES) which will be used when copying subband samples from the source channel to the current channel ch. These scale factors are encoded in exactly the same way as that for SCALES. Use Table 5-24 to look up the codebook.

JOIN_SCALES (Scale Factors for Joint Subband Coding)

The scale factors are used to scale the subband samples copied from the source channel (JOINX[ch]-1) to the current channel. The index of the scale factor is encoded using the code book indexed by JOIN_SHUFF[ch]. After this index is decoded, it is used to look up the table in clause D.3 to get the scale factor. No transient is permitted for jointly encoded subbands, so a single scale factor is included. The joint subbands start from the nSUBS of the current channel until the nSUBS of the source channel.

SICRC (Side Information CRC Check Word)

If CPF = 1 then SICRC shall be extracted from the bitstream. The CRC value test shall not be applied.

6.4.4.2 Data Arrays

Table 6-20: XCH audio data arrays

XCH Data Arrays	Size (Bits)

for (ch=0; ch<nPCHS; ch++)
  for (n=nVQSUB[ch]; n<nSUBS[ch]; n++) {
  // Extract the VQ address from the bit stream
  nVQIndex = ExtractBits(10);
  // Look up the VQ code book for 32 subband samples.
  HFreqVQ.LookUp(nVQIndex, HFREQ[ch][n]);
  // Scale and take the samples
  rScale = (real)SCALES[ch][n][0]; // Get the scale factor
  for (m=0; m<nSSC*8; m++, nSample++)
  aPrmCh[ch].aSubband[n].raSample[m] = rScale*HFREQ[ch][n][m];
	10 bits per applicable sub band

Audio Data	

// Select quantization step size table
if (RATE == 0x1f)
  pStepSizeTable = &StepSizeLossLess; // Lossless quantization
 else
  pStepSizeTable = &StepSizeLossy; // Lossy
// Unpack the subband samples
for (nSub-sub-frame=0; nSub-sub-frame<nSSC; nSub-sub-frame++) {
  for (ch=0; ch<nPCHS; ch++)
  for (n=0; n<nVQSUB[ch]; n++) { // Not high frequency VQ subbands
  //
  // Select the mid-tread linear quantizer
  //
  nABITS = ABITS[ch][n]; // Select the mid-tread quantizer
  pCQGroup = &pCQGroupAUDIO[nABITS-1];// Select the group of
  // code books corresponding to the
  // the mid-tread linear quantizer.
  nNumQ = pCQGroupAUDIO[nABITS-1].nNumQ-1;// Number of code
  // books in this group
  //
  // Determine quantization index code book and its type
  //
	variable bits

XCH Data Arrays	Size (Bits)

```
    // Select quantization index code book
    nSEL = SEL[ch][nABITS-1];
    // Determine its type
    nQType = 1;   // Assume Huffman type by default
    if ( nSEL==nNumQ ) {  // Not Huffman type
if ( nABITS<=7 )
  nQType = 3; // Block code
else
  nQType = 2; // No further encoding
    }
    if ( nABITS==0 )  // No bits allocated
nQType = 0;
    //
    // Extract bits from the bit stream
    //
    switch ( nQType ) {
    case 0 :  // No bits allocated
for (m=0; m<8; m++)
  AUDIO[m] = 0;
break;
    case 1 :  // Huffman code
for (m=0; m<8; m++)
  pCQGroup->ppQ[nSEL]->InverseQ(InputFrame,AUDIO[m]);
break;
    case 2 :  // No further encoding
for (m=0; m<8; m++) {
  // Extract quantization index from the bit stream
  pCQGroup->ppQ[nSEL]->InverseQ(InputFrame, nCode)
    // Take care of 2's compliment
      AUDIO[m] = pCQGroup->ppQ[nSEL]->SignExtension(nCode);
}
break;
    case 3 : // Block code
pCBQ = &pCBlockQ[nABITS-1]; // Select block code book
m = 0;
for (nBlock=0; nBlock<2; nBlock++) {
  // Extract the block code index from the bit stream
  pCQGroup->ppQ[nSEL]->InverseQ(InputFrame, nCode)
    // Look up 4 samples from the block code book
    pCBQ->LookUp(nCode,&AUDIO[m])
    m += 4;
}
break;
    default: // Undefined
print ("ERROR: Unknown AUDIO quantization index code book.");
    }
  }
            // Account for quantization step size and scale factor
// Look up quantization step size
nABITS = ABITS[ch][n];
pStepSizeTable->LookUp(nABITS, rStepSize);
// Identify transient location
nTmode = TMODE[ch][n];
if ( nTmode == 0 ) // No transient
  nTmode = nSSC;
// Determine proper scale factor
if (nSub-sub-frame<nTmode)        // Pre-transient
  rScale = rStepSize * SCALES[ch][n][0];  // Use first scale factor
else  // After-transient
  rScale = rStepSize * SCALES[ch][n][1];  // Use second scale factor
// Adjustmemt of scale factor
rScale *= arADJ[ch][SEL[ch][nABITS-1]];   // arADJ[ ][ ] are assumed 1
// unless changed by bit
// stream when SEL indicates
// Huffman code.
// Scale the samples
nSample = 8*nSub-sub-frame;       // Set sample index
for (m=0; m<8; m++, nSample++)
  aPrmCh[ch].aSubband[n].aSample[nSample] = rScale*AUDIO[m];
// Inverse ADPCM
if ( PMODE[ch][n] != 0 ) // Only when prediction mode is on.
  aPrmCh[ch].aSubband[n].InverseADPCM();
// Check for DSYNC
if ( (nSub-sub-frame==(nSSC-1)) || (ASPF==1) ) {
  DSYNC = ExtractBits(16);
  if ( DSYNC != 0xffff )
    printf("DSYNC error at end of sub-sub-frame #%d", nSub-sub-frame);
```

XCH Data Arrays	Size (Bits)
} } }	

HFREQ (VQ Encoded High Frequency Subbands)

At low bit rates, some high frequency subbands are encoded using vector quantization (VQ). The code book is given in clause D.10.2. Each vector from this code book consists of 32 subband samples, corresponding to the maximum possible subframe (4 normal subsubframes):

4 subsubframe × 8 samples/subsubframe = 32 samples

If the current subframe is short of 32 samples, the remaining samples are padded with zeros and then vector-quantized. The vector address is then included in the bit stream. After the decoder picks up the vector address, it looks up the vector code book to get the 32 samples. But the decoder will only pick nSSC×8 out of the 32 samples and scale them with the scale factor SCALES.

AUDIO (Audio Data)

The audio data are grouped as nSSC subsubframes, each consisting of 8 samples for each subband. Each sample was quantized by a mid-tread linear quantizer indexed by ABITS. The resultant quantization index may further be encoded by either a Huffman or block code. If it is not, it is included in the bit stream as 2's compliment. All this information is indicated by SEL. The (ABITS, SEL) pair then tells how the subband samples should be extracted from the bit stream (Table 5-26).

The resultant subband samples are then compensated by their respective quantization step sizes and scale factors. Special care is to be paid to possible transient in the subframe. If a transient is flagged by TMODE, one scale factor will be used for samples before the transient and the other one for the after the transient.

For some of the subbands that are ADPCM encoded, the samples of these subbands thus far obtained are actually the difference signals. Their real values are recovered through a reverse ADPCM process:

- At end of each subsubframe there may be a synchronization check word DSYNC = 0xffff depending on the flag ASPF in the frame header, but there shall be at least a DSYNC at the end of each subframe.

6.5 Extension to More Than 5.1 Channels (XXCH)

6.5.1 About the XXCH Extension

The XXCH extension supports lossy encoding of more than 5.1 channels by combining the backward compatible core (of up to 5.1 channels) with up to 32 additional channels. The extended channels are compressed using the same technology as the core audio channels. The audio data representing these extension channels may be included either into a core substream or into an extension substream. The XXCH data within the core substream is automatically ignored by all DTS legacy decoders but can be decoded by DTS-HD decoders.

The frame of XXCH data is divided into a frame header and up to four channel sets. Each channel set has its own channel set header as shown in Figure 6-11.The CRC word is included at the end of frame header to allow detection of errors in the frame header data. In addition, the CRC words may be included at the end of each channel set header to allow detection of errors in the channel set header data.

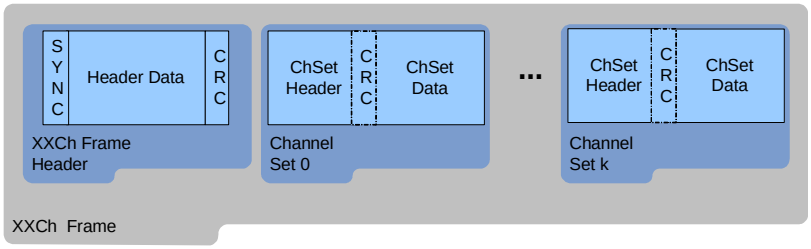


Figure 6-11: XXCH Data Frame Structure

The channel set data is furthermore subdivided into the subframes. Each subframe consists of a subframe header and the audio data. The audio data is organized in subsubframes as shown in Figure 6-12. The number of subframes is the same for all channel sets and is equal to the number of subframes in the core frame. Similarly the number of subsubframes is the same for all subframes of all channel sets and is equal to the number of subsubframes within each subframe of the core frame. In other words the subframe and subsubframe partitioning within the XXCH frame follows the partitioning present in the core frame.

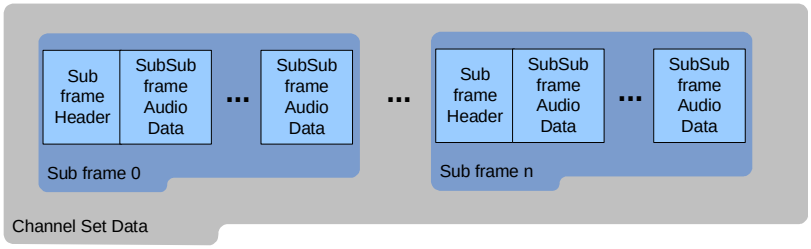


Figure 6-12: Channel Set Data Structure

6.5.2 XXCH Frame Header

Table 6-21: XXCH Frame Header Structure

XXCH Frame Header Syntax	Size (Bits)
`SYNCXXCh = ExtractBits(32);`	32
`nuHeaderSizeXXCh = ExtractBits(6)+1;`	6
`bCRCPresent4ChSetHeaderXXCh = (ExtractBits(1)==1) ? true : false;`	1
`nuBits4SpkrMaskXXCh = ExtractBits(5)+1;`	5
`nuNumChSetsInXXCh = ExtractBits(2)+1;`	2
`for (nChSet=0; nChSet < nuNumChSetsInXXCh; nChSet ++)` `  pnuChSetFsizeXXCh[nChSet] = ExtractBits(14)+1;`	14*nuNumChSetsInXXCh
`nuCoreSpkrActivityMask = ExtractBits(nuBits4SpkrMaskXXCh);`	nuBits4SpkrMaskXXCh
`ReservedHeaderXXCh = ExtractBits(…);`	…
`ByteAlignHeaderXXCh = ExtractBits(0 … 7);`	0...7
`nCRC16HeaderXXCh = ExtractBits(16);`	16

SYNCXXCh (XXCH Sync Word)

The DWORD aligned XXCH synchronization word has value 0x47004a03.

nuHeaderSizeXXCh (XXCH frame header length)

nuHeaderSizeXXCh is the size of the header in bytes from the SYNCXXCh to the nCRC16HeaderXXCh inclusive. This value determines the location of the first channel set header. This marker also designates the end of the field nCRC16HeaderXXCh and allows quick location of the checksum at byte position nuHeaderSizeXXCh-2.

bCRCPresent4ChSetHeaderXXCh (CRC presence flag for channel set header)

When bCRCPresent4ChSetHeaderXXCh = true the 16-bit CRC word for the channel set header is present at the end of each channel set header.

nuBits4SpkrMaskXXCh (Number of bits for loudspeaker mask)

nuBits4SpkrMaskXXCh indicates how many bits are used for packing the loudspeaker layout mask nuXXChSpkrLayoutMask (packed using nuBits4SpkrMaskXXCh - 6 bits) and the nDownMixChMapMask (packed using nuBits4SpkrMaskXXCh bits).

nuNumChSetsInXXCh (Number of Channel Sets)

All channels within the XXCH extension are organized in individually decodable channel sets. The nNumChSetaInXXCh is the number of channel sets that are present in XXCH.

pnuChSetFsizeXXCh (Channel Set Data Byte Size)

The pnuChSetFsizeXXCh[nChSet] indicates the total number of data bytes in nChSet channel set of the XXCH frame. Starting from the SYNCXXCh and using the cumulative sum of nuHeaderSizeXXCh and the pnuChSetFsizeXXCh[k] (over all channel sets k=0, … nChSet -1) as offset, decoder may traverse to the beginning of channel set header data in the channel set nChSet.

nuCoreSpkrActivityMask (Core Loudspeaker Activity Mask)

The nuCoreSpkrActivityMask indicates which of the pre-defined loudspeaker positions apply to the audio channels encoded in the core portion of DTS-HD stream. In nominal case the core channel layout as indicated by the AMODE core parameter will agree with the layout indicated by the nuCoreSpkrActivityMask. However there may be cases where the two layouts are not identical and:

- the information in the AMODE field of the core stream is used to determine the intended speaker layout for 5.1 playback;
- the information in nuCoreSpkrActivityMask and nuXXChSpkrLayoutMask when combined together is used to determine the intended speaker layout for 8 channel playback.

As an example consider the channel layout, that is coded in DTS-HD stream, to be C, L, R, LFE_1, L_{ss}, R_{ss}, L_{sr} and R_{sr}. In this case channels are organized in two channels sets with the C, L, R, LFE_1, L_{ss}, R_{ss}, being in the first (core) channel set and the L_{sr} and R_{sr} being in the second (XXCH) channel set. During the encode process the 7.1 to 5.1 down-mix will be embedded such that the L_{sr} and R_{sr} channels are mixed into the L_{ss} and R_{ss} channels. The resulting mixed channels are encoded in the core stream as Ls and Rs channels (AMODE=9 $\geq$ C, L, R, Ls and Rs layout). A 5.1 decoder uses this AMODE to configure its decoded outputs to C, L, R, L_s and R_s layout. On the other hand a 7.1 decoder ignores the AMODE information from the core stream and uses instead the nuCoreSpkrActivityMask (C, L, R, LFE_1, L_{ss} and R_{ss}) and the nuXXChSpkrLayoutMask (L_{sr} and R_{sr}) from the XXCh stream to get the original 7.1 speaker layout (C, L, R, LFE_1, L_{ss}, R_{sr}, L_{sr} and R_{sr}) and configures its outputs accordingly.

Each core encoded channel or channel pair, depending on the corresponding speaker position(s), sets the appropriate bit in a loudspeaker activity mask. Predetermined loudspeaker positions are described in Table 6-22. For example, nuSpkrActivityMask = 0xF indicates activity of C, L, R, L_s, R_s and LFE_1 loudspeakers.

Table 6-22: Loudspeaker Masks - nuXXChSpkrLayoutMask/nuCoreSpkrActivityMask/DownMixChMapMask

Notation	Location Description (Approximate angle in horizontal plane)	Corresponding bit in nuXXChSpkrLayoutMask	Corresponding bit in nuCoreSpkrActivityMask/ DownMixChMapMask
C	Centre in front of listener (0)	N/A	0x00000001
L	Left in front (-30)	N/A	0x00000002
R	Right in front (30)	N/A	0x00000004
Ls	Left surround on side in rear (-110)	N/A	0x00000008
Rs	Right surround on side in rear (110)	N/A	0x00000010
LFE1	Low frequency effects subwoofer	N/A	0x00000020
Cs	Centre surround in rear (180)	0x00000040	0x00000040
LSr	Left surround in rear (-150)	0x00000080	0x00000080
Rsr	Right surround in rear (150)	0x00000100	0x00000100
Lss	Left surround on side (-90)	0x00000200	0x00000200
Rss	Right surround on side (90)	0x00000400	0x00000400
Lc	Between left and centre in front (-15)	0x00000800	0x00000800
Rc	Between right and centre in front (15)	0x00001000	0x00001000
Lh	Left height in front	0x00002000	0x00002000
Ch	Centre Height in front	0x00004000	0x00004000
Rh	Right Height in front	0x00008000	0x00008000
LFE2	Second low frequency effects subwoofer	0x00010000	0x00010000
Lw	Left on side in front (-60)	0x00020000	0x00020000
Rw	Right on side in front (60)	0x00040000	0x00040000
Oh	Over the listener's head	0x00080000	0x00080000
Lhs	Left height on side	0x00100000	0x00100000
Rhs	Right height on side	0x00200000	0x00200000
Chr	Centre height in rear	0x00400000	0x00400000
Lhr	Left height in rear	0x00800000	0x00800000
Rhr	Right height in rear	0x01000000	0x01000000
Cl	Centre in the plane lower then listener's ears	0x02000000	0x02000000
Ll	Left in the plane lower then listener's ears	0x04000000	0x04000000
Rl	Right in the plane lower then listener's ears	0x08000000	0x08000000
Reserved		0x10000000 to 0x80000000	0x10000000 to 0x80000000

ReservedHeaderXXCh (Reserved)

This field is reserved for additional XXCH header information. The decoder shall assume that this field is present and of unspecified duration. Therefore, in order to continue unpacking the stream, the decoder shall skip over this field using the XXCH header start pointer and the XXCH header size nuHeaderSizeXXCh.

ByteAlignHeaderXXCh (Pad to BYTE boundary)

This field ensures that the CRC16 field that follows is aligned to a byte boundary to allow fast table based CRC16 calculation. Append '0's until bit position is a multiple of 8.

nCRC16HeaderXXCh (CRC16 of XXCH frame header)

It represents the 16-bit CRC check word of the entire XXCH header from position nuHeaderSizeXXCh to ByteAlignHeaderXXCh inclusive.

6.5.3 XXCH Channel Set Header

Table 6-23: XXCh Channel Set Header Structure

XXCH Channel Set Header Syntax	Size (Bits)
nuXXChChSetHeaderSize = ExtractBits(7)+1;	7
nuChInChSetXXCh= ExtractBits(3) + 1;	3
nuXXChSpkrLayoutMask = ExtractBits(nuBits4SpkrMaskXXCh-6) << 6;	nuBits4SpkrMaskXXCh-6
bDownMixCoeffCodeEmbedded = (ExtractBits(1) == 1) ? true : false;	1
if (bDownMixCoeffCodeEmbedded){	
bDownMixEmbedded = (ExtractBits(1) == 1) ? true : false;	1
nDmixScaleFactor = ExtractBits(6);	6
for (nCh=0; nCh<nuChInChSet; nCh++) DownMixChMapMask[nCh] = ExtractBits(nuBits4SpkrMaskXXCh);	nuBits4SpkrMaskXXCh
for (nCh=0; nCh<nuChInChSet; nCh++){ for (n=0, nCoef=0; n<nuBits4SpkrMaskXXCh; n++, nCoef++){ if ((DownMixChMapMask[nCh]>>n) & 0x1) DownMixCoeffs[nCh][nCoef] = ExtractBits(7); } }	7
} // End condition on bDownMixCoeffCodeEmbedded	
for (nCh=0; nCh<nuChInChSet; nCh++) SUBS[nCh] = ExtractBits(5);	5
for (nCh=0; nCh<nuChInChSet; nCh++) VQSUB[nCh] = ExtractBits(5);	5
for (nCh=0; nCh<nuChInChSet; nCh++) JOINX[nCh] = ExtractBits(3);	3
for (nCh=0; nCh<nuChInChSet; nCh++) THUFF[nCh] = ExtractBits(2);	2
for (nCh=0; nCh<nuChInChSet; nCh++) SHUFF[nCh] = ExtractBits(3);	3
for (nCh=0; nCh<nuChInChSet; nCh++) BHUFF[nCh] = ExtractBits(3);	3

// ABITS=1:
n=0;
for (ch=0; ch<nuChInChSetXXCh; ch++)
SEL[ch][n] = ExtractBits(1);
// ABITS = 2 to 5:
for (n=1; n<5; n++)
for (ch=0; ch<nuChInChSetXXCh; ch++)
SEL[ch][n] = ExtractBits(2);
// ABITS = 6 to 10:
for (n=5; n<10; n++)
for (ch=0; ch<nuChInChSetXXCh; ch++)
SEL[ch][n] = ExtractBits(3);
// ABITS = 11 to 26:
for (n=10; n<26; n++)
for (ch=0; ch<nuChInChSetXXCh; ch++)
SEL[ch][n] = 0; // Not transmitted, set to zero.
// ABITS = 1 :
n = 0;
for (ch=0; ch<nuChInChSetXXCh; ch++){
if (SEL[ch][n] == 0)
arADJ[ch][n] = AdjTable[ExtractBits(2)];
}
// ABITS = 2 to 5:
for (n=1; n<5; n++){
for (ch=0; ch<nuChInChSetXXCh; ch++){
if (SEL[ch][n] < 3)
arADJ[ch][n] = AdjTable[ExtractBits(2)];
}
// ABITS = 6 to 10:
for (n=5; n<10; n++){
for (ch=0; ch<nuChInChSetXXCh; ch++)
if (SEL[ch][n] < 7)
arADJ[ch][n] = AdjTable[ExtractBits(2)];
}
	Variable bits
	2

ReservedHeaderChSet = ExtractBits(…);	…
ByteAlignHeaderChSet = ExtractBits(0 … 7);	0...7
if (bCRCPresent4ChSeHeader) nCRC16HeaderChSet = ExtractBits(16);	16

nuXXChChSetHeaderSize (Channel set header length)

The size of the channel set header in bytes from the nuXXChChSetHeaderSize to either ByteAlignHeaderChSet (when bCRCPresent4ChSetHeaderXXCh = false) or nCRC16HeaderChSet (when bCRCPresent4ChSetHeaderXXCh = true) inclusive. This value determines the beginning of the channel set audio data. In case when the bCRCPresent4ChSetHeaderXXCh = true this marker also designates the end of the field nCRC16HeaderChSet and allows quick location of the checksum at byte position nuXXChChSetHeaderSize -2.

nuChInChSetXXCh (Number of channels in a channel set)

Indicates the number of channels in the channel set.

nuXXChSpkrLayoutMask (Loudspeaker Layout Mask)

The nuXXChSpkrLayoutMask indicates which of the pre-defined loudspeaker positions apply to the audio channels encoded in a XXCh channel set. Each encoded channel/channel pair, depending on the corresponding speaker position/positions, sets the appropriate bit in a loudspeaker layout mask. Predetermined loudspeaker positions are described in Table 6-22. For example nuXXChSpkrLayoutMask =0x4040 indicates activity of Cs and Ch loudspeakers. This is in addition to the speakers encoded in the core portion of the DTS-HD stream.

bDownMixCoeffCodeEmbedded (Downmix coefficients present in stream)

If true it indicates that a matrix of downmix coefficients has been defined and is embedded in the stream.

bDownMixEmbedded (Downmix already performed by encoder)

Present only if bDownMixCoeffEmbedded is true. When bDownMixEmbedded=true this indicates to the decoder that on the encode side, audio in the channels of the current channel set (nChSet) have been down mixed to the core channels and to the channels in the lower indexed channel sets (ChSetIndex<nChSet). After decoding the current channel set the above mentioned encoder downmix operation needs to be undone in the decoder. If bDownMixEmbedded=false the encoder did not perform the downmixing operation on the current channel set.

nDmixScaleFactor (Downmix scale factor)

Present only if bDownMixCoeffEmbedded is true. The nDmixScaleFactor is a scaling coefficient that prevents an overflow and is applied to all output channels of the downmix. In the case when bDownMixCoeffEmbedded is true the nDmixScaleFactor has already been applied, on the encode side, to the output downmix channels.

DownMixChMapMask (Downmix channel mapping mask)

Present only if bDownMixCoeffEmbedded is true. Each channel of the current channel set may be mapped to any of the core and the channels in the lower indexed channel sets. For each channel (nCh) of the current channel set, the DownMixChMapMask[nCh] indicates the core channels and the channels in the lower indexed channel sets to which the channel nCh is going to be downmixed into.

The DownMixChMapMask[nCh] has a dedicated bit (according to Table 6-22) for each of the core channels and each of the channels in the lower indexed channels sets. A channel nCh of the current channel set is mapped exclusively to those channels that have their bits in DownMixChMapMask [nCh] set to "1". Mapping to all other channels assumes downmix coefficient equal to -Infinity.

DownMixCoeffs (Downmix coefficients)

Present only if bDownMixCoeffEmbedded is true. For each channel (nCh) of the current channel set there is one downmix coefficient for each bit that is set to "1" in the corresponding channel mapping mask DownMixChMapMask[nCh]. These coefficients are used to multiply each sample of the channel nCh and add the product to the corresponding sample of the channel indicated by the DownMixChMapMask[nCh].

Coding of the downmix coefficients is described in clause D.11.

SUBS (Subband Activity Count)

This field indicates that there are nSUBS[ch] = SUBS[ch]+2 active subbands in the audio channel ch. Samples in subbands above nSUBS[ch] are zero, provided that intensity coding in that subband is disabled.

VQSUB (High Frequency VQ Start Subband)

This field indicates that high frequency samples starting from subband nVQSUB[ch]=VQSUB[ch]+1 are VQ encoded. High frequency VQ is used only for high frequency subbands, but it may go down to low frequency subbands for such audio episodes as silence. In case of insufficient MIPS, the VQs for the highest frequency subbands may be ignored without causing audible distortion.

JOINX (Joint Intensity Coding Index)

This field indicates if joint intensity coding is enabled for channel ch and which audio channel is the source channel from which channel ch will copy subband samples (see Table 5-22). It is assumed that the source channel index is smaller than that of the intensity channel.

THUFF (Transient Mode Code Book)

This field indicates which Huffman codebook was used to encode the transient mode data TMODE (see Table 5-23).

SHUFF (Scale Factor Code Book)

The scale factors of a channel are quantized nonlinearly using either a 6-bit (64-level, 2,2 dB per step) or a 7-bit (128-level, 1,1 dB per step) square root` table, depending on the application. The quantization indexes may be further compressed by one of the five Huffman codes and this information is transmitted to the decoder by SHUFF[ch] (see Table 5-24).

BHUFF (Bit Allocation Quantizer Select)

This field indicates the codebook that was used to encode the bit allocation index ABITS (see Table 5-25).

SEL (Quantization Index Codebook Select)

After subband samples are quantized using a mid-tread linear quantizer, the quantization indexes may be further encoded using either entropy (Huffman) or block coding in order to reduce bit rate. Therefore, the subband samples may appear in the bitstream as plain quantization indexes (no further encoding), entropy (Huffman) codes, or block codes. For channel ch, the selection of a particular codebook for a mid-tread linear quantizer indexed by ABITS[ch] is transmitted to the decoder as SEL[ch][ABITS[ch]]. No SEL is transmitted for ABITS[ch]≥11, because no further encoding is used for those quantizers. The decoder can find out the particular codebook that was used using ABITS[ch] and SEL[ch][ABITS[ch]] to look up the table (see Table 5-26).

ADJ (Scale Factor Adjustment Index)

A scale factor adjustment index is transmitted whenever a SEL value indicates a Huffman codebook. This index points to the adjustment values shown in the table (see Table 5-27). This adjustment value should be multiplied by the scale factor (SCALE).

ReservedHeaderChSet (Reserved)

This field is reserved for additional channel set header information. The decoder shall assume that this field is present and of unspecified length. Therefore, in order to continue unpacking the stream, the decoder shall skip over this field using the channel set header start pointer and the channel set header size nuXXChChSetHeaderSize.

ByteAlignHeaderChSet (Pad to BYTE boundary)

This field ensures that the CRC16 field that follows is aligned to a byte boundary to allow fast table based CRC16 calculation. Append '0's until bit position is a multiple of 8.

nCRC16HeaderChSet (CRC16 of channel set header)

This field is present only if the bCRCPresent4ChSetHeaderXXCh is true. It represents the 16-bit CRC check word of the entire channel set header from positions nuXXChChSetHeaderSize to ByteAlignHeaderChSet inclusive.

6.5.4 Unpack Subframes

6.5.4.1 Unpack Subframes Syntax

Table 6-24: XXCH Unpack Subframes

XXCH Unpack Subframes	Size (Bits)

`for (ch=0; ch<nuChInChSetXXCh; ch++)`
`  for (n=0; n<nSUBS[ch]; n++)`
`    PMODE[ch][n] = ExtractBits(1);`
	1 bit per subband

`int nVQIndex;`
`for (ch=0; ch<nuChInChSetXXCh; ch++) {`
`  for (n=0; n<nSUBS[ch]; n++)`
`    if ( PMODE[ch][n]>0 ) { // Transmitted only when ADPCM active`
`      // Extract the VQindex`
`      nVQIndex = ExtractBits(12);`
`      // Look up the VQ table for prediction coefficients.`
`      ADPCMCoeffVQ.LookUp(nVQIndex, PVQ[ch][n]); // 4 coefficients`
`    }`
`}`
	12 bits per occurrence

`for (ch=0; ch<nuChInChSetXXCh; ch++) {`
`  // BHUFF tells which codebook was used`
`  nQSelect = BHUFF[ch];`
`  // Use this codebook to decode the bitstream for ABITS[ch][n]`
`  for (n=0; n<nVQSUB[ch]; n++) // Not for VQ encoded subbands.`
`    QABITS.ppQ[nQSelect]->InverseQ(InputFrame, ABITS[ch][n]);`
`}`
	Variable bits

`// Always assume no transition unless told`
`for (ch=0; ch<nuChInChSetXXCh; ch++)`
`  for (n=0; n<NumSubband; n++)`
`  TMODE[ch][n] = 0;`
`// Decode TMODE[ch][n]`
`if ( nSSC>1 ) {// Transient possible only if more than one sub-sub-frame.`
`  for (ch=0; ch<nuChInChSetXXCh; ch++) {`
`// TMODE[ch][n] is encoded by a codebook indexed by THUFF[ch]`
`nQSelect = THUFF[ch];`
`for (n=0; n<nVQSUB[ch]; n++) // No VQ encoded subbands`
`  if ( ABITS[ch][n] >0 ) { // Present only if bits allocated`
`  // Use codebook nQSelect to decode TMODE from the bitstream`
`     QTMODE.ppQ[nQSelect]->InverseQ(InputFrame,TMODE[ch][n])`
	Variable bits

`Scale Factors`
`for (ch=0; ch<nuChInChSetXXCh; ch++) {`
`  // Clear SCALES`
`  for (n=0; n<NumSubband; n++) {`
`       SCALES[ch][n][0] = 0;`
`    SCALES[ch][n][1] = 0;`
`  }`
`  // SHUFF indicates which codebook was used to encode SCALES`
`  nQSelect = SHUFF[ch];`
`  // Select the root square table (SCALES were nonlinearly`
`  // quantized).`
`  if ( nQSelect == 6 )`
`    pScaleTable = &RMS7Bit; // 7-bit root square table`
`  else`
`    pScaleTable = &RMS6Bit; // 6-bit root square table`
`  //`
`  // Clear accumulation (if Huffman code was used, the difference`
`  // of SCALES was encoded).`
`  //`
`  nScaleSum = 0;`
`  //`
`  // Extract SCALES for Subbands up to VQSUB[ch]`
`  //`
`  for (n=0; n<nVQSUB[ch]; n++)`
`    if ( ABITS[ch][n] >0 ) { // Not present if no bit allocated`
`      //`
`      // First scale factor`
`      //`
`      // Use the (Huffman) code indicated by nQSelect to decode`
`      // the quantization index of SCALES from the bitstream`
`      QSCALES.ppQ[nQSelect]->InverseQ(InputFrame, nScale);`
`      // Take care of difference encoding`
`      if ( nQSelect < 5 )   // Huffman encoded, nScale is the difference`
`  nScaleSum += nScale;  // of the quantization indexes of SCALES.`
	Variable bits

XXCH Unpack Subframes	Size (Bits)

`      else // Otherwise, nScale is the quantization`
`  nScaleSum = nScale;   // level of SCALES.`
`      // Look up SCALES from the root square table`
`      pScaleTable->LookUp(nScaleSum, SCALES[ch][n][0])`
`  //`
`  // Two scale factors transmitted if there is a transient`
`  //`
`  if (TMODE[ch][n]>0) {`
`    // Use the (Huffman) code indicated by nQSelect to decode`
`    // the quantization index of SCALES from the bitstream`
`    QSCALES.ppQ[nQSelect]->InverseQ(InputFrame, nScale);`
`    // Take care of difference encoding`
`    if ( nQSelect < 5 ) // Huffman encoded, nScale is the`
`      nScaleSum += nScale;  // of SCALES.`
`    else    // Otherwise, nScale is SCALES`
`      nScaleSum = nScale;   // itself.`
`    // Look up SCALES from the root square table`
`    pScaleTable->LookUp(nScaleSum, SCALES[ch][n][1]);`
`  }`
`    }`
`  //`
`  // High frequency VQ subbands`
`  //`
`  for (n=nVQSUB[ch]; n<nSUBS[ch]; n++) {`
`    // Use the code book indicated by nQSelect to decode`
`    // the quantization index of SCALES from the bitstream`
`    QSCALES.ppQ[nQSelect]->InverseQ(InputFrame, nScale);`
`    // Take care of difference encoding`
`    if ( nQSelect < 5 )     // Huffman encoded, nScale is the`
`      nScaleSum += nScale;  // of SCALES.`
`    else  // Otherwise, nScale is SCALES`
`      nScaleSum = nScale;   // itself.`
`    // Look up SCALES from the root square table`
`    pScaleTable->LookUp(nScaleSum, SCALES[ch][n][0]);`
`  }`
 	
`for (ch=0; ch<nuChInChSetXXCh; ch++)`
`  if ( JOINX[ch]>0 ) // Transmitted only if joint subband coding enabled.`
`    JOIN_SHUFF[ch] = ExtractBits(3);`
`int nSourceCh;`
`for (ch=0; ch<nuChInChSetXXCh; ch++) {`
`  if ( JOINX[ch]>0 ) { // Only if joint subband coding enabled.`
`    nSourceCh = JOINX[ch]-1;    // Get source channel. JOINX counts`
`    // channels as 1,2,3,4,5, so minus 1.`
`    nQSelect = JOIN_SHUFF[ch]; // Select code book.`
`    for (n=nSUBS[ch]; n<nSUBS[nSourceCh]; n++) {`
`      // Use the code book indicated by nQSelect to decode`
`      // the quantization index of JOIN_SCALES`
`      QSCALES.ppQ[nQSelect]->InverseQ(InputFrame, nJScale);`
`      // Bias by 64`
`      nJScale = nJScale + 64;`
`      // Look up JOIN_SCALES from the joint scale table`
`      JScaleTbl.LookUp(nJScale, JOIN_SCALES[ch][n]);`
`    }`
`  }`
`if ( CPF==1 ) // Present only if CPF=1.`
`  SICRC = ExtractBits(16);`
	3 bits per channel
	
	Variable bits

6.5.4.2 Side Information

Prediction Mode (PMODE)

PMODE[ch][n]=1 (1 bit per subband) indicates that ADPCM prediction is used (active) for subband n of extension audio channel [ch] and PMODE[ch][n]=0 otherwise. ADPCM shall be extracted from the bitstream for all subbands, but ADPCM reconstruction can be limited to the lowest 20 subbands if DSP does not have enough MIPS.

Prediction Coefficients VQ Address (PVQ)

This field (12 bits per active occurrence) indexes to the vector code book to get the ADPCM prediction coefficients, (see clause D.10.2). It is transmitted only for subbands whose ADPCM is active.

Bit Allocation Index (ABITS)

ABITS[ch][n] (variable bits) is the index to the mid-tread linear quantizer that was used to quantize the subband samples for the n^{th} subband of channel ch. ABITS[ch][n] may be transmitted as either a 4-bit or 5-bit word. When ABITS is encoded in a 4-bit word, it may be further encoded using one of the five Huffman codes. This encoding is the same for all subbands of each channel and is conveyed by BHUFF as shown in Table 5-26. There is no need to allocate bits for the high frequency subbands because they are encoded using VQ.

Transition Mode (TMODE)

TMODE[ch][n] (variable bits) indicates if there is a transient inside a subframe (subband analysis window) for subband n of channel ch. If there is a transient (TMODE[ch][n]>0), it further indicates that the transition occurred in subsubframe (subband analysis subwindow) TMODE[ch][n] + 1. TMODE[ch][n] is encoded by one of the four Huffman codes and the selection of which is conveyed by THUFF (see Table 6-7). The decoder assumes that there is no transition (TMODE[ch][n]=0) for all subbands of all channels unless it is told otherwise by the bitstream. Transient does not occur in the following situations, so TMODE is not transmitted:

- Only one subsubframe within the current subframe. This is because the time resolution of transient analysis is a subsubframe (subband analysis subwindow).
- VQ encoded high frequency subbands. If there is a transient for a subband, it would not have been VQ encoded.
- Subbands without bit allocation. If there is no need to allocate bits for a subband, there is no need to care about transient for it.

Scale Factors (SCALES)

One scale factor (variable bits) is transmitted for subbands without transient. Otherwise two are transmitted, one for the episode before the transient and the other for after the transient. The quantization indexes of the scale factors may be encoded by Huffman code as shown in Table 5-24. If this is the case, they are difference-encoded before Huffman coding. The scale factors are finally obtained by using the quantization indexes to look up either the 6-bit or 7-bit square root quantization table according to Table 5-24.

Joint Subband Scale Factor Codebook Select (JOIN SHUFF)

If joint subband coding is enabled (JOINX[ch]>0), JOIN_SHUFF[ch] (3 bits per channel) selects which code book was used to encode the scale factors (JOIN_SCALES) which will be used when copying subband samples from the source channel to the current channel ch. For now, these scale factors are encoded in exactly the same way as that for SCALES, so use Table 5-24 to look up the codebook.

Scale Factors for Joint Subband Coding (JOIN SCALES)

The scale factors (variable bits) are used to scale the subband samples copied from the source channel (JOINX[ch]-1) to the current channel. The index of the scale factor is encoded using the code book indexed by JOIN SHUFF[ch]. After this index is decoded, it is used to look up the table in clause D.3 in to get the scale factor. No transient is permitted for jointly encoded subbands, so a single scale factor is included. The joint subbands start from the nSUBS of the current channel until the nSUBS of the source channel.

Side Information CRC Check Word (SICRC)

If CPF = 1 then SICRC shall be extracted from the bitstream. The CRC value test shall not be applied.

6.5.4.3 Data Arrays

Table 6-25: XXCH - Data Arrays

XXCH Data Arrays	Size (Bits)

for (ch=0; ch<nPCHS; ch++)
  for (n=nVQSUB[ch]; n<nSUBS[ch]; n++) {
  // Extract the VQ address from the bitstream
  nVQIndex = ExtractBits(10);
  // Look up the VQ code book for 32 subband samples.
  HFreqVQ.LookUp(nVQIndex, HFREQ[ch][n])
  // Scale and take the samples
  Scale = (real)SCALES[ch][n][0]; // Get the scale factor
  for (m=0; m<nSSC*8; m++, nSample++)
  aPrmCh[ch].aSubband[n].raSample[m] = rScale*HFREQ[ch][n][m];
  }
	10 bits per subband

AUDIO Section	**Variable bits**

// Select quantization step size table
if (RATE == 0x1f)
  pStepSizeTable = &StepSizeLossLess; // Lossless quantization
 else
  pStepSizeTable = &StepSizeLossy; // Lossy
//
// Unpack the subband samples
for (nSub-sub-frame=0; nSub-sub-frame<nSSC; nSub-sub-frame++) {
  for (ch=0; ch<nPCHS; ch++)
  for (n=0; n<nVQSUB[ch]; n++) { // Not high frequency VQ subbands
  //
  // Select the mid-tread linear quantizer
  //
  nABITS = ABITS[ch][n]; // Select the mid-tread quantizer
  pCQGroup = &pCQGroupAUDIO[nABITS-1];// Select the group of
  // code books corresponding to the
  // the mid-tread linear quantizer.
  nNumQ = pCQGroupAUDIO[nABITS-1].nNumQ-1;// Number of code
  // books in this group
  //
  // Determine quantization index code book and its type
  //
  // Select quantization index code book
  nSEL = SEL[ch][nABITS-1];
  // Determine its type
  nQType = 1; // Assume Huffman type by default
  if (nSEL==nNumQ) { // Not Huffman type
  if (nABITS<=7)
  nQType = 3; // Block code
  else
  nQType = 2; // No further encoding
  }
  if (nABITS==0) // No bits allocated
  nQType = 0;
  //
  // Extract bits from the bitstream
  //
  switch (nQType) {
  case 0 : // No bits allocated
  for (m=0; m<8; m++)
  AUDIO[m] = 0;
  break;
  case 1 : // Huffman code
  for (m=0; m<8; m++)
  pCQGroup->ppQ[nSEL]->InverseQ(InputFrame,AUDIO[m]);
  break;
  case 2 : // No further encoding
  for (m=0; m<8; m++) {
  // Extract quantization index from the bitstream
  pCQGroup->ppQ[nSEL]->InverseQ(InputFrame, nCode)
  // Take care of 2's compliment
  AUDIO[m] = pCQGroup->ppQ[nSEL]->SignExtension(nCode);
  }
  break;
  case 3 : // Block code
  pCBQ = &pCBlockQ[nABITS-1]; // Select block code book
  m = 0;
  for (nBlock=0; nBlock<2; nBlock++) {
  // Extract the block code index from the bitstream	

```
    pCQGroup->ppQ[nSEL]->InverseQ(InputFrame, nCode)
      // Look up 4 samples from the block code book
      pCBQ->LookUp(nCode,&AUDIO[m])
      m += 4;
  }
  break;
      default: // Undefined
  printf("ERROR: Unknown AUDIO quantization index code book.");
      }
    }
  //
  // Account for quantization step size and scale factor
  //
  // Look up quantization step size
  nABITS = ABITS[ch][n];
  pStepSizeTable->LookUp(nABITS, rStepSize);
  // Identify transient location
  nTmode = TMODE[ch][n];
  if ( nTmode == 0 ) // No transient
    nTmode = nSSC;
  // Determine proper scale factor
  if (nSub-sub-frame<nTmode) // Pre-transient
    rScale = rStepSize * SCALES[ch][n][0]; // Use first scale factor
  else  // After-transient
    rScale = rStepSize * SCALES[ch][n][1]; // Use second scale factor
  // Adjustmemt of scale factor
  rScale *= arADJ[ch][SEL[ch][nABITS-1]];   // arADJ[ ][ ] are assumed 1
  // unless changed by bit
  // stream when SEL indicates
  // Huffman code.
  // Scale the samples
  nSample = 8*nSub-sub-frame; // Set sample index
  for (m=0; m<8; m++, nSample++)
    aPrmCh[ch].aSubband[n].aSample[nSample] = rScale*AUDIO[m];
  //
  // Inverse ADPCM
  //
  if ( PMODE[ch][n] != 0 ) // Only when prediction mode is on.
    aPrmCh[ch].aSubband[n].InverseADPCM();
  //
  // Check for DSYNC
  if ( (nSub-sub-frame==(nSSC-1)) || (ASPF==1) ) {
    DSYNC = ExtractBits(16);
    if ( DSYNC != 0xffff )
      printf("DSYNC error at end of sub-sub-frame #%d", nSub-sub-frame);
  }
 }
}
```

VQ Encoded High Frequency Subbands (HFREQ)

At low bit rates, some high frequency subbands are encoded using vector quantization (VQ). The code book is given in clause D.10.2. Each vector from this code book consists of 32 subband samples, corresponding to the maximum possible subframe (4 normal subsubframes):

$$4 \text{ subsubframe} \times 8 \text{ samples/subsubframe} = 32 \text{ samples}$$

If the current subframe is short of 32 samples, the remaining samples are padded with zeros and then vector- s, it looks up the vector code book to get the 32 samples. But the decoder will only pick nSSC×8 out of the 32 samples and scale them with the scale factor SCALES.

Audio Data (AUDIO)

The audio data are grouped as nSSC subsubframes, each consisting of eight samples for each subband. Each sample was quantized by a mid-tread linear quantizer indexed by ABITS. The resultant quantization index may be further encoded by either a Huffman or block code. If it is not, it is included in the bitstream as 2's compliment. All this information is indicated by SEL. The (ABITS, SEL) pair then tells how the subband samples should be extracted from the bitstream (see Table 6-8).

The resultant subband samples are then compensated by their respective quantization step sizes and scale factors. Special care is to be paid to possible transient in the subframe. If a transient is flagged by TMODE, one scale factor is used for samples before the transient and the other one for the after the transient.

For some of the subbands that are ADPCM encoded, the samples of these subbands thus far obtained are actually the difference signals. Their real values shall be recovered through a reverse ADPCM process:

At the end of each subsubframe there may be a synchronization check word DSYNC = 0xffff depending on the flag ASPF in the frame header, but there shall be at least a DSYNC at the end of each subframe.

ReservedChSet (Reserved)

This field is reserved for additional channel set information. The decoder shall assume that this field is present and of unspecified duration. In order to continue unpacking the stream, the decoder shall skip over this field by navigating **nuXXChChSetHeaderSize** bytes from the beginning of the XXCH header.

ByteAlignChSet (Pad to BYTE boundary)

This field insures that the XXCH extension ends on a byte boundary. 0 to 7 bits which are set to 0 are added to force byte boundary alignment.

7 DTS Extension Substream Construction

7.1 About the DTS Extension Substream

Building upon the foundation established by the original DTS codec technology, the DTS extension substream formats have added a modular architecture to include support for advanced features such as alternate channel maps, the ability to include replacement channels and metadata to permit authored control in the play back environment of how these streams are selected or combined. Additionally, the ability to support resolution enhancements and a new coding profile designed for higher efficiency, bring together a new comprehensive suite of technologies into one package.

This expanded coding system, of which an implementation is currently commercialized as DTS-HD™, can support compatibility with pre-existing audio decoding systems through the presence of the core substream, enhancing performance of the core substream, or attain higher efficiency or a combination of efficiency and performance when direct compatibility with the original DTS core is not required.

7.2 Relationship Between Core and Extension Substreams

Organization into the core substream and the extension substreams is illustrated in Figure 7-1.

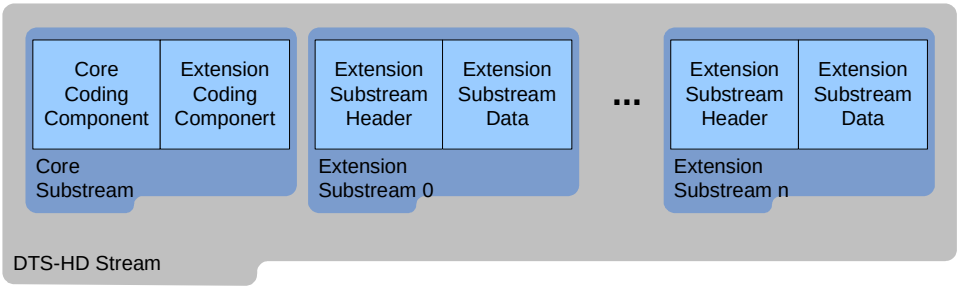


Figure 7-1: Organization of the DTS-HD stream

The core substream carries a DTS Coherent Acoustics component which is referred to as the core and may carry one extension as indicated in clause 5.

An extension substream may consist of one or more of these components:

- Coding components that represent the extensions to the core coding component present in the core substream. The use of these extensions enhances an audio presentation by providing features such as higher resolution and additional channels.
- Additional audio assets that are mixed with the audio asset represented by the core substream and/or other assets from the extension substreams, creating a new audio presentation.

- Stand-alone audio assets that represent an entire audio presentation.

When compatibility to previously deployed DTS decoders is not required, the audio stream may consist of one or more extension substreams.

7.3 Audio Presentations and Audio Assets

7.3.1 Overview of Extension Substream Architecture

An extension substream has an architecture that allows several different audio presentations to be coded within a single extension substream. At the first level, each extension substream is organized in up to eight audio assets. Some examples are:

- An integral part of an audio presentation that is mixed with another audio asset to create the complete audio presentation. For example, one audio asset may carry the music, effects and dialog while another asset may carry a director's commentary. The complete audio presentation is prepared by mixing the director's commentary asset with the music, effects and dialog asset. The instructions for mixing are transmitted in the associated metadata fields.

- A set of loudspeaker signal feeds for a corresponding loudspeaker layout, such as, feed for left, right, left surround, right surround, centre and LFE loudspeakers in 5.1 loudspeaker layout.

- A set of signals that describe the sound field but are not actual loudspeaker feeds. The actual loudspeaker feeds are derived from these signals using stream-embedded metadata and/or some signal processing, i.e. Ambisonic B-format where signals W, X, Y and Z are transmitted and the actual loudspeaker feeds are derived on the decoder side using linear mixing equations.

Note that by default the core substream data belongs to the asset 0.

The layout of an extension substream is illustrated in Figure 7-2.

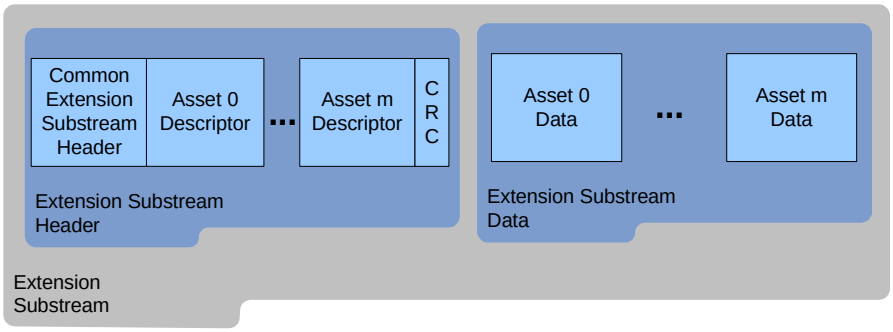


Figure 7-2: Organization of the Extension Substream

It is not necessary that all of the audio assets present in an audio stream be active at the same time. In particular, different audio presentations are defined by activating specific audio assets. The active assets are combined together as instructed by the stream metadata to create a particular audio presentation. For example:

- Audio asset 0 carries the primary audio presentation.

- Audio asset 1 carries a director's commentary in English.

- Audio asset 2 carries a director's commentary in Spanish.

With this configuration of audio assets, one can define the following three audio presentations:

- Activate the asset 0 to create the primary audio presentation.

- Activate the assets 0 and 1 to create the primary audio presentation mixed with the director's commentary in English.

- Activate the assets 0 and 2 to create the primary audio presentation mixed with the director's commentary in Spanish.

Furthermore, an audio presentation may consist of the active assets that are transmitted within different extension substreams. In particular, each extension substream can define up to eight different audio presentations. An audio presentation that is defined in the extension substream with index nExSS may include any audio asset from all of the extension substreams with an index less than or equal to nExSS.

For example:

- Extension substream with index 0:
  - Audio asset 0 carries primary audio presentation.
  - Audio asset 1 carries a director's commentary in English.
- Extension substream with index 1:
  - Audio asset 0 carries a director's commentary in Spanish.

With the configuration of extension substreams and the audio assets within each of them (shown above), the following audio presentations can be created:

- Audio presentations defined in extension substream 0:
  - Primary audio presentation by activating the asset 0 of extension substream 0.
  - Primary audio + Secondary English by activating the assets 0 and 1 of extension substream 0.
- Audio presentations defined in extension substream 1:
  - Primary audio + Secondary Spanish by activating the asset 0 of extension substream 0 and asset 0 of extension substream 1.

Note that different coding components of asset 0 may be either in the core substream or in the extension substream. All assets other than asset 0 shall have all their coding components within the same extension substream.

The organization of an audio asset data is illustrated in Figure 7-3.

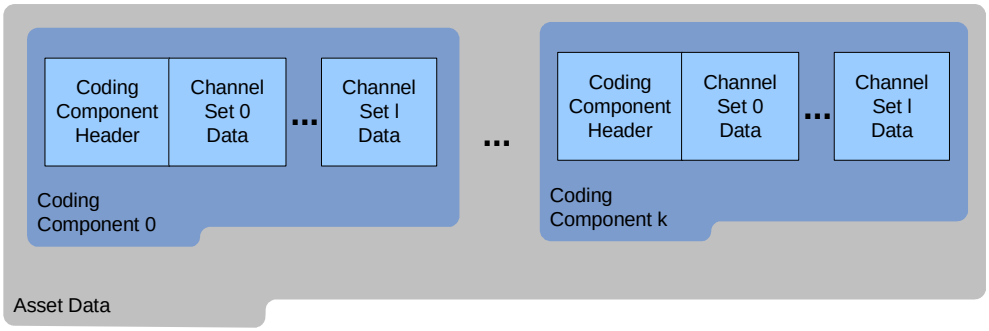


Figure 7-3: Organization of the Audio Asset Data

7.3.2 Channel Sets

In order to achieve scalability, the architecture of the extension substream allows the channels within the coding components to be organized into channel sets. Each channel set can be separately extracted and decoded as needed. An example of this would be when the intended speaker layout for a channel set, e.g. a 7.1 mix can be encoded in three channel sets such that all three of the following apply:

- Decoding of a channel set 0 produces a stereo downmix.
- Decoding and combining of the channel sets 0 and 1 produces a 5.1 downmix.
- Decoding and combining of the channel sets 0, 1 and 2 produces the original 7.1 mix.

7.4 Synchronization and Navigation of the Substream

7.4.1 Synchronization

The list of possible sync words for core and substream components is provided in Table 7-1.

Table 7-1: Sync Words

DTS_SYNCWORD_CORE	0x7ffe8001
DTS_SYNCWORD_XCH	0x5a5a5a5a
DTS_SYNCWORD_XXCH	0x47004a03
DTS_SYNCWORD_X96	0x1d95f262
DTS_SYNCWORD_XBR	0x655e315e
DTS_SYNCWORD_LBR	0x0a801921
DTS_SYNCWORD_XLL	0x41a29547
DTS_SYNCWORD_SUBSTREAM	0x64582025
DTS_SYNCWORD_SUBSTREAM_CORE	0x02b09261

DTS_SYNCWORD_SUBSTREAM_CORE, which is located in the extension substream, has a sync word that is remapped from 0x7ffe8001 to 0x02b09261. This makes the core substream synchronization simpler and more robust. When this backward compatible core component is to be delivered to a legacy DTS decoders, (e.g. via SPDIF), its sync word needs to be restored from 0x02b09261 to 0x7ffe8001 prior to the transmission to the legacy decoders.

7.4.2 Substream Navigation

Parsing of the substream data depends on the error-free determination of the size (FSIZE) of each component. This FSIZE value, when accumulated, helps rapidly locate the start of each component. To ensure error free location, the FSIZE values in the substream header are protected by a checksum and, at a minimum, the decoder should attempt to decode the legacy core component to ensure some minimal level of audio output.

Navigation through the individual components within the substream is achieved by extracting and accumulating the FSIZE of each component. An index table is gradually constructed allowing a decoder to rapidly locate the start of any data stream. After offsetting the pointer from the present position, the checksum field is immediately found and the data present at that location is verified for integrity before any further parsing and processing, as shown in Figure 7-4.

If a component fails a checksum field-and if the substream header was intact-the index table allows the decoder to locate the next useful packet of data. However, interdependency between the data may be such that any extra data can no longer be utilized. At a minimum, the decoder should always attempt to decode the legacy core where present and augment it with relevant component data.

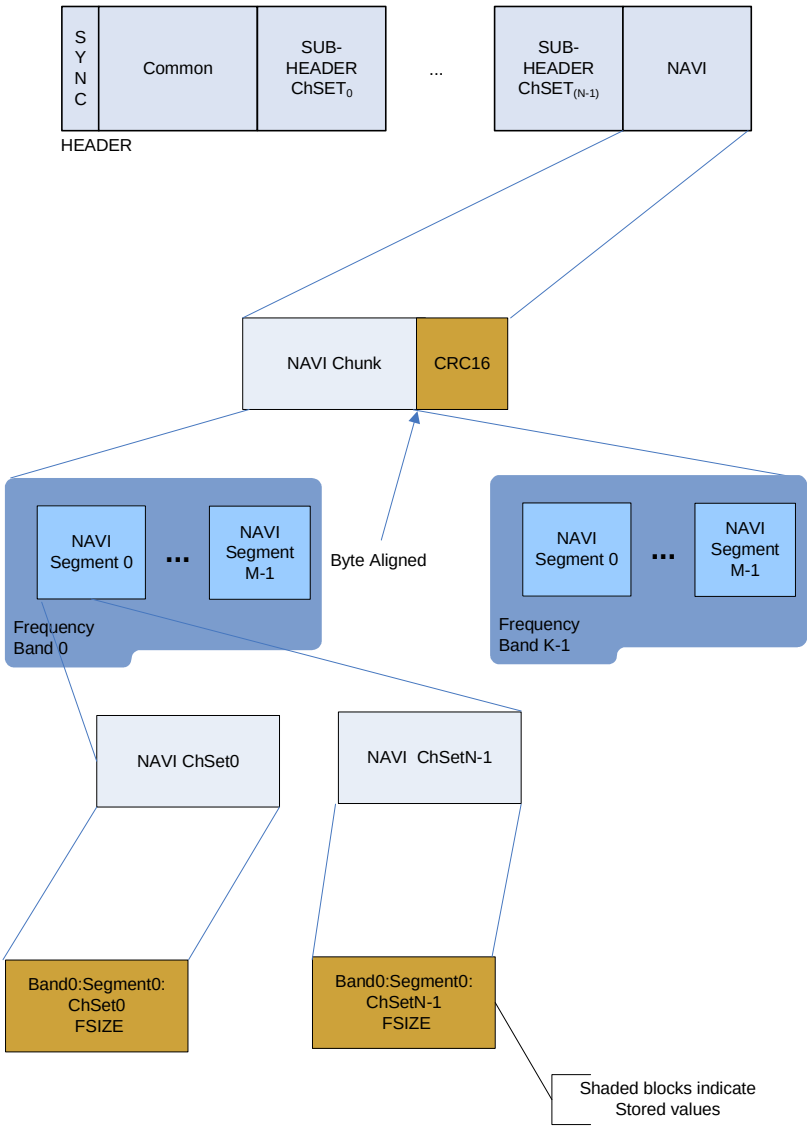


Figure 7-4: NAVI Synchronization

7.5 Parsing Core Substream and Extension Substream Data

7.5.1 General Information on Parsing Substreams

Core substream and extension substream(s) need to be composed in a specific manner when being presented to a decoder, as depicted in Figure 7-5.

When the data is sent to the decoder, both the sync words that define the start of the legacy core substream and the extension substream shall be aligned to 32-bit boundaries. The length of the core substream is byte-aligned, so the system layer may need to introduce from one to three null bytes between the core substream and the extension substream. If any additional substream, such as from an external file, is sent to the decoder, proper SYNC word alignment shall be maintained.

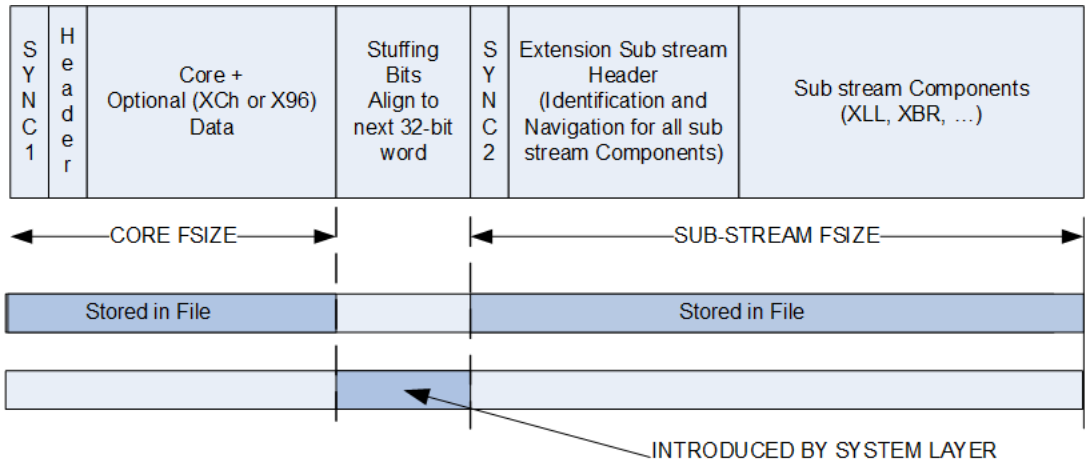


Figure 7-5: DWORD alignment of substream

7.5.2 Extension Substream Header

The extension substream header describes the audio assets that are present in the extension substream and the instructions for manipulating the various assets. The syntax of the extension substream header is described in Table 7-2, which follows.

Table 7-2: Extension Substream Header Structure

Extension Substream Header Structure	Size (Bits)
`// Extract the sync word`	
`SYNCEXTSSH = ExtractBits (32);`	32
`UserDefinedBits = ExtractBits(8);`	8
`nExtSSIndex = ExtractBits(2);`	2

`// Unpack the num of bits to be used to read header size`
`bHeaderSizeType = ExtractBits(1);`
`if (bHeaderSizeType == 0){`
`  nuBits4Header = 8;`
`  nuBits4ExSSFsize = 16`
`}`
`else{`
`  nuBits4Header = 12;`
`  nuBits4ExSSFsize = 20;`
`}`
	1

`// Unpack the substream header size`	
`nuExtSSHeaderSize = ExtractBits(nuBits4Header) + 1;`	nuBits4Header
`nuExtSSFsize = ExtractBits(nuBits4ExSSFsize) + 1;`	nuBits4ExSSFsize
`bStaticFieldsPresent = ExtractBits(1);`	1
`if (bStaticFieldsPresent){`	
`  nuRefClockCode = ExtractBits(2);`	2
`  nuExSSFrameDurationCode = 512*(ExtractBits(3)+1);`	3
`  bTimeStampFlag = ExtractBits(1);`	1

`  if (bTimeStampFlag)`
`  {`
`  nuTimeStamp = ExtractBits(32);`
`  nLSB = ExtractBits(4);`
`  nuTimeStamp = ((DTS__int64) (nuTimeStamp<<4)) | nLSB;`
`  }`
	36

`  nuNumAudioPresnt = ExtractBits(3)+1;`	3
`  nuNumAssets = ExtractBits(3)+1;`	3
`  for (nAuPr=0; nAuPr<nuNumAudioPresnt; nAuPr++)` `  nuActiveExSSMask[nAuPr] = ExtractBits(nExtSSIndex+1);`	nExtSSIndex+1

`  for (nAuPr=0; nAuPr<nuNumAudioPresnt; nAuPr++){`
`  for (nSS=0; nSS<nExtSSIndex+1; nSS++){`
`  if (((nuActiveExSSMask[nAuPr]>>nSS) & 0x1) == 1)`
`  nuActiveAssetMask[nAuPr][nSS] = ExtractBits(8);`
`  else`
	8

Extension Substream Header Structure	Size (Bits)

nuActiveAssetMask[nAuPr][nSS]= 0;
}
}
bMixMetadataEnbl = ExtractBits(1);
if (bMixMetadataEnbl){
nuMixMetadataAdjLevel = ExtractBits(2);
nuBits4MixOutMask = (ExtractBits(2)+1)<<2;
nuNumMixOutConfigs = ExtractBits(2) + 1;
// Output Mixing Configuration Loop
for (ns=0; ns<nuNumMixOutConfigs; ns++){
nuMixOutChMask[ns]= ExtractBits(nBits4MixOutMask);
nNumMixOutCh[ns] =
NumSpkrTableLookUp(nuMixOutChMask[ns]);
}
} // End of if (bMixMetadataEnbl)
} // End of if (bStaticFieldsPresent)
else // bStaticFieldsPresent==false
{
nuNumAudioPresnt = 1;
nuNumAssets = 1;
}
for (nAst=0; nAst< nuNumAssets; nAst++)
nuAssetFsize[nAst] = ExtractBits(nuBits4ExSSFsize)+1;
for (nAst=0; nAst< nuNumAssets; nAst++)
AssetDescriptor{}
for (nAuPr=0; nAuPr<nuNumAudioPresnt; nAuPr++)
bBcCorePresent[nAuPr] = ExtractBits(1);
for (nAuPr=0; nAuPr<nuNumAudioPresnt; nAuPr++){
if (bBcCorePresent[nAuPr])
nuBcCoreExtSSIndex[nAuPr] = ExtractBits(2);
nuBcCoreAssetIndex[nAuPr]= ExtractBits(3);
}
Reserved = ExtractBits(…);
ByteAlign = ExtractBits(0 … 7);
nCRC16ExtSSHeader = ExtractBits(16);
	1
	2
	2
	2
	nuBits4MixOutMask
	nuBits4ExSSFsize×nuNumAssets
	See Asset Descriptor in Table 7-5.
	1
	2
	3
	…
	0...7
	16

SYNCEXTSSH (Extension Substream Sync Word)

The extension substream has a DWORD-aligned synchronization word with the hexadecimal value of 0x64582025. During sync detection, the nCRC16Header checksum (see Annex B) is used to further verify that the detected sync pattern is not a random alias.

UserDefinedBits (User Defined Field)

This field is reserved and may be used by an encoder operator. This field is not included in the Metadata CRC check and as such can be freely altered post encoding. This field represents the beginning of the metadata block and it is assumed that its start location within the encoded bit-stream is at the byte boundary.

nExtSSIndex (Extension Substream Index)

It is possible to have up to four extension substreams, originating from different sources, e.g. disc, Ethernet, broadcast, hard drive, etc., that are concatenated one after another. The nExtSSIndex parameter indicates an index of each extension substream and helps the decoder to differentiate between the different extension substreams. Its range is from 0 to 3.

bHeaderSizeType (Flag Indicating Short or Long Header Size)

If bHeaderSizeType is 0, the header size is short (up to 256 Bytes) and is expressed using 8 bits. If the bHeaderSizeType is 1, the header size is long (up to 4 kBytes) and is expressed using 12 bits.

nuExtSSHeaderSize (Extension Substream Header Length)

This is the size of the extension substream header in bytes from the SYNCEXTSSH to nCRC16ExtSSHeader inclusive. This value determines the location of the first component of the first audio asset of the extension substream. This marker also designates the end of the field nCRC16ExtSSHeader and makes it possible to quickly locate the checksum at byte position nuExtSSHeaderSize-2.

nuExtSSFsize (Number of Bytes of Extension Substream)

This is the number of bytes in the current frame of extension substream. This value is used to traverse to the end of the extension substream frame.

bStaticFieldsPresent (Per Stream Static Fields Presence Flag)

If this field is true, it indicates that the current frame includes the metadata fields that are static over the duration of an encoded stream. If the bStaticFieldsPresent is false, the metadata fields that are static over the duration of an encoded stream are omitted from the extension substream header. In this case, decoders should set the number of assets to 1 and the number of audio presentations to 1.

nuRefClockCode (Reference Clock Code)

This field indicates the reference clock period. The reference clock period is used for calculating a frame duration and a decoder presentation time of an extension substream, as shown below in Table 7-3. The reference clock period (RefClockPeriod) is calculated from the extracted unsigned integer 2-bit field (nuRefClockCode) using a look-up table, as shown in Table 7-3.

Table 7-3: Reference Clock Period

nuRefClockCode	RefClockPeriod [seconds]
0	1,0 / 32 000,0
1	1,0 / 44 100,0
2	1,0 / 48 000,0
3	Unused

nuExSSFrameDurationCode (Extension Substream Frame Duration)

This field indicates the time duration between the two consecutive occurrences of the extension substream header. This duration is expressed by the number of clock cycles using the reference clock indicated by the value in RefClockPeriod. The number of clock cycles (nuExSSFrameDurationCode) is derived from the extracted unsigned integer 3-bit field by multiplying its value by 512. The actual duration in seconds (ExSSFrameDuration)is calculated from the nuExSSFrameDurationCode and the RefClockPeriod in the following manner:

$$ExSSFrameDuration = nuExSSFrameDurationCode \times RefClockPeriod.$$

bTimeStampFlag (Timecode presence Flag)

This is present only if bStaticFieldsPresent is true. When bTimeStampFlag = 1, the time code field is present.

nuTimeStamp (Timecode data)

This is present only if bStaticFieldsPresent and bTimeStampFlag are true. The timestamp data is a 36-bit field composed as follows:

$$nuTimeStamp = \frac{Hours \times 3600 + Mins \times 60 + Sec}{\text{RefClockPeriod}} + SampleOffset$$

Where all of the following are true:

- The Hours has range 0 to 23
- The Mins has range 0 to 59
- The Sec has range 0 to 59
- The 1/ RefClockPeriod may be 32 000, 44 100, or 48 000, as deduced from Table 7-3
- The SampleOffset has range of 0 to 31 999, 44 099, or 47 999 respectively

The timestamp of an encoded frame (N) corresponds to that time when the first edge of the first bit of the encoded frame (N) shall be clocked to the decoder; i.e. it is decoder presentation time for frame (N).

nuNumAudioPresnt (Number of Defined Audio Presentations)

This is present only if bStaticFieldsPresent is true. It indicates the number of audio presentations that are defined in this extension substream. The audio presentations are defined in terms of active extension substreams-which may have an index from 0 to nExtSSIndex-and active assets within each active extension substream. From one to eight audio presentations may be defined.

nuNumAssets (Number of Audio Assets in Extension Substream)

This is present only if bStaticFieldsPresent is true. It indicates the total number of audio assets that are encoded in an extension substream. An extension substream may consist of up to eight audio assets.

nuActiveExSSMask (Active Extension Substream Mask for an Audio Presentation)

This is present only if bStaticFieldsPresent is true. The audio assets from different extension substreams may be combined together to create an audio presentation. The location of "1" bits in nuActiveExSSMask indicates the indices of the active extension substreams that are used to create an audio presentation. If more than two extension substreams are to be combined together, the process of combining them is performed in stages, starting from the lowest index assets in the lowest index extension substream. Only the current and the lower indexed extension substreams may be combined into an audio presentation defined in the current extension substream.

nuActiveAssetMask (Active Audio Asset Mask)

This is present only if bStaticFieldsPresent is true. The audio assets from all active substreams are combined together to create an audio presentation. The location of "1" bits in nuActiveAssetMask[nAuPr][nSS] indicates the indices of audio assets in the extension substream with index nSS that are used to create the audio presentation with index nAuPr.

If more than two audio assets are to be combined together, the process of combining them is performed in stages, starting from the lowest index assets.

bMixMetadataEnbl (Mixing Metadata Enable Flag)

This is present only if bStaticFieldsPresent is true. This field is true if at least one of the audio assets present in this substream extension is encoded for mixing or replacement. If bMixMetadataEnbl is false, all audio assets of this substream extension are encoded as standalone audio presentations.

nuMixMetadataAdjLevel (Mixing Metadata Adjustment Level)

This is present only if bStaticFieldsPresent and bMixMetadataEnbl are true. Both the system metadata, which includes the listener's preferences and the bit-stream metadata, can be present in a mixing audio application. The audio asset mixing features that can be adjusted by system metadata are:

- Feature 1: The level of an audio asset, relative to the level of other audio assets involved in mixing.
- Feature 2: The placement of an audio asset within the sound field by means of altering the mixing coefficients.

The mixing metadata adjustment level (nuMixMetadataAdjLevel) specifies which feature(s) (1 and/or 2 above), if any, may be adjusted, as described in Table 7-4. nuMixMetadataAdjLevel=3 is reserved for an additional level of adjustment that may be specified in the future. If nuMixMetadataAdjLevel=3, all decoders shall assume that, at minimum, the adjustment of features 1 and 2 is allowed.

Table 7-4: Allowed Mixing Metadata Adjustment Level

NuMixMetadataAdjLevel	Metadata Usage
0	Use only bitstream metadata
1	Allow system metadata to adjust feature 1
2	Allow system metadata to adjust both feature 1 and feature 2
3	Reserved

nuBits4MixOutMask (Number of Bits for Mixer Output Speaker Activity Mask)

This is present only if both bStaticFieldsPresent and bMixMetadataEnbl are true. Its value indicates how many bits are used for packing a mixer output speaker activity mask nuMixOutChMask. Valid values for nuBits4MixOutMask are 4, 8, 12 and 16 and are obtained from the extracted 2-bit code using the following mapping:

$$nuBits4MixOutMask = (code+1)<<2.$$

nuNumMixOutConfigs (Number of Mixing Configurations)

This is present only if both bStaticFieldsPresent and bMixMetadataEnbl are true. nuNumMixOutConfigs indicates the number of metadata sets (each corresponding to a different mixer output speaker configuration) included in the metadata of each mixing audio asset.

The content provider can transmit mixing metadata that controls the mixing for several speaker configurations that can be extracted from the main audio stream. For example, within a 10.2 channels main audio presentation, it is likely that 5.1 and 7.1 downmixes are already embedded in the main audio stream. In this case, the content provider may choose to provide two or even three sets of mixing metadata to do one or more of the following:

- Control mixing of the 5.1 main audio asset downmix with the supplemental audio asset.
- Control mixing of the 7.1 main audio asset downmix with the supplemental audio asset.
- Control mixing of the 10.2 main audio asset with the supplemental audio asset.

A mixer in its minimal implementation may perform the mixing outlined in case 1, above, followed by the downmix to 2 channels. However an advanced player may be able to support the mixing outlined in case 2 or 3, above.

nuMixOutChMask (Speaker Layout Mask for Mixer Output Channels)

This is present only if both bStaticFieldsPresent and bMixMetadataEnbl are true. This field defines the channel layout for mixer output channels. This information is needed to associate the mixing coefficients with appropriate channels. (See description of a nuSpkrActivityMask in Table 7-10.)

nuAssetFsize (Size of Encoded Asset Data in Bytes)

This is the number of bytes for an encoded audio asset in current frame. The audio asset descriptor metadata is not included in nuAssetFsize.

In particular, the beginning of data that is included in the nuAssetFsize[0] for audio asset 0 is at the offset of nuExtSSHeaderSize bytes from the beginning of the extension substream header. The beginning of encoded data for audio asset 1 is at the offset of nuExtSSHeaderSize+nuAssetFsize[0] bytes, etc.

Notice that when an audio asset's data is split between the core substream and the extension substream, the nuAssetFsize does not include the core data.

AssetDescriptor (Audio Asset Descriptor)

This is the audio asset descriptor is present for each encoded audio asset. Each audio asset may represent different audio asset types, such as Music and Effects, Dialog and Commentary. Each audio asset may be encoded with a different number of channels, a different sampling frequency and a different bit-width. The audio asset descriptor is a block of metadata that describes each of these parameters. Details are described in clause 7.5.3.

bBcCorePresent (Backward Compatible Core Present)

By default, the core substream is sent over SPDIF to guarantee backward compatibility. However, in the case of a stream with multiple presentations, an alternate audio presentation may have a backward compatible core component that resides in an extension substream. When the bBcCorePresent flag is true for a particular audio presentation, the backward compatible core is present in the stream. In this case, the field nuBcCoreExtSSIndex and the field nuBcCoreAssetIndex indicate, respectively, the index of an extension substream and the index of an audio asset that contains the backward compatible core. By default the core sub-stream data, if present, has the nuBcCoreExtSSIndex = 0 and the nuBcCoreAssetIndex = 0. Based on the active presentation, the internal decoder shall determine which backward compatible core shall be parsed out of the DTS-HD stream and delivered via SPDIF to the backward compatible decoder. If the backward compatible core has been extracted from the extension sub-stream then prior to sending it via SPDIF the internal decoder shall replace its existing sync word by the value of 0x7FFE8001 (backward compatible sync word). The amount of data that is to be transmitted via SPDIF is determined from:

- the core frame header parameter FSIZE, for the case when the backward compatible core is obtained from the core sub-stream;
- the asset descriptor parameter nuExSSCoreFsize parameter, for the case when the backward compatible core is obtained from the extension sub-stream.

nuBcCoreExtSSIndex (Backward Compatible Core Extension Substream Index)

This field is present only when bBcCorePresent is true. This field indicates the index of the extension substream that contains an asset with a backward compatible core for a specific audio presentation.

nuBcCoreAssetIndex (Backward Compatible Core Asset Index)

This field is present only when bBcCorePresent is true. This field indicates the index of an audio asset that contains the backward compatible core for a specific audio presentation.

Reserved (Reserved)

This field is reserved for additional extension substream header information. The decoder shall assume that this field is present and of unspecified duration. Therefore in order to continue unpacking the stream, the decoder shall skip over this field using the extension substream header start pointer and the extension substream header size nuExtSSHeaderSize.

nCRC16ExtSSHeader (CRC16 of Extension Substream Header)

nCRC16ExtSSHeader is the CRC16 of the entire extension substream header from positions nExtSSIndex to ByteAlign, inclusive. See Annex B for details of the CRC algorithm used.

7.5.3 Audio Asset Descriptor

7.5.3.1 General Information About the Audio Asset Descriptor

The audio asset descriptor is present for each encoded audio asset. The audio asset descriptor is a block of metadata that describes each of these parameters. Its syntax is provided in Table 7-5, Table 7-6 and Table 7-7.

Note that a new pseudo-function, 'unsigned int CountBitsSet_to_1(unsigned int *nuWord*)' is introduced, first appearing in Table 7-5. The function calculates the number of bits that are set to "1" in the parameter *nuWord*.

Table 7-5: Audio Asset Descriptor Syntax: Size, Index and Per Stream Static Metadata

Audio asset Descriptor Syntax	Size (Bits)
`nuAssetDescriptFsize = ExtractBits(9)+1;`	9
`nuAssetIndex = ExtractBits(3);`	3
`if (bStaticFieldsPresent){` `  bAssetTypeDescrPresent = ExtractBits(1);`	1
`  if (bAssetTypeDescrPresent)` `     nuAssetTypeDescriptor = ExtractBits(4);`	4
`  bLanguageDescrPresent = ExtractBits(1);`	1
`  if (bLanguageDescrPresent)` `     LanguageDescriptor = ExtractBits(24);`	24

Audio asset Descriptor Syntax	Size (Bits)
bInfoTextPresent = ExtractBits(1);	1
if (bInfoTextPresent) nuInfoTextByteSize = ExtractBits(10)+1;	10
if (bInfoTextPresent) InfoTextString = ExtractBits(nuInfoTextByteSize*8);	nuInfoTextByteSize×8
nuBitResolution = ExtractBits(5) + 1;	5
nuMaxSampleRate = ExtractBits(4)	4
nuTotalNumChs = ExtractBits(8)+1;	8
bOne2OneMapChannels2Speakers = ExtractBits(1);	1
if (bOne2OneMapChannels2Speakers){	

if (nuTotalNumChs>2)
bEmbeddedStereoFlag = ExtractBits(1);
else
bEmbeddedStereoFlag = 0;
	1

if (nuTotalNumChs>6)
bEmbeddedSixChFlag = ExtractBits(1);
else
bEmbeddedSixChFlag = 0;
	1

bSpkrMaskEnabled = ExtractBits(1);	1
if (bSpkrMaskEnabled) nuNumBits4SAMask = (ExtractBits(2)+1)<<2;	2
if (bSpkrMaskEnabled) nuSpkrActivityMask = ExtractBits(nuNumBits4SAMask);	nuNumBits4SAMask
nuNumSpkrRemapSets = ExtractBits(3);	3
for (ns=0; ns<nuNumSpkrRemapSets; ns++) nuStndrSpkrLayoutMask[ns] = ExtractBits(nuNumBits4SAMask);	nuNumBits4SAMask

for (ns=0;ns<nuNumSpkrRemapSets; ns++){
nuNumSpeakers = NumSpkrTableLookUp(nuStndrSpkrLayoutMask[ns]);
nuNumDecCh4Remap[ns] = ExtractBits(5)+1;
	5

for (nCh=0; nCh<nuNumSpeakers; nCh++){ // Output channel loop
nuRemapDecChMask[ns][nCh] = ExtractBits(nuNumDecCh4Remap[ns]);
nCoef = CountBitsSet_to_1(nuRemapDecChMask[ns][nCh]);
for (nc=0; nc<nCoef; nc++)
	nuNumDecCh4Remap[ns]

nuSpkrRemapCodes[ns][nCh][nc] = ExtractBits(5);	5
} // End output channel loop } // End nuNumSpkrRemapSets loop	
} // End of if (bOne2OneMapChannels2Speakers)	

else{ // No speaker feed case
bEmbeddedStereoFlag = false;
bEmbeddedSixChFlag = false;
nuRepresentationType = ExtractBits(3);
	3

} } // End of if (bStaticFieldsPresent)	

Table 7-6: Audio Asset Descriptor Syntax:
Dynamic Metadata - DRC, DNC and Mixing Metadata

Audio asset Descriptor Syntax	Size (Bits)
bDRCCoefPresent = ExtractBits(1);	1
if (bDRCCoefPresent) nuDRCCode = ExtractBits(8);	8
bDialNormPresent = ExtractBits(1);	1
if (bDialNormPresent) nuDialNormCode = ExtractBits(5);	5
if (bDRCCoefPresent && bEmbeddedStereoFlag) nuDRC2ChDmixCode = ExtractBits(8);	8

if (bMixMetadataEnbl)
bMixMetadataPresent = ExtractBits(1);
else
bMixMetadataPresent = false;
	1

if (bMixMetadataPresent){	
bExternalMixFlag = ExtractBits(1);	1
nuPostMixGainAdjCode = ExtractBits(6);	6
nuControlMixerDRC = ExtractBits(2);	2
if (nuControlMixerDRC <3) nuLimit4EmbeddedDRC = ExtractBits(3);	3
if (nuControlMixerDRC ==3) nuCustomDRCCode = ExtractBits(8);	8
bEnblPerChMainAudioScale = ExtractBits(1);	1

for (ns=0; ns<nuNumMixOutConfigs; ns++){
if (bEnblPerChMainAudioScale){
for (nCh=0; nCh<nNumMixOutCh[ns]; nCh++)
	6

Audio asset Descriptor Syntax	Size (Bits)
`      nuMainAudioScaleCode[ns][nCh] = ExtractBits(6);`	
`    }`	
`    else`	
`      nuMainAudioScaleCode[ns][0] = ExtractBits(6);`	
`  }`	
`  nEmDM = 1;`	
`  nDecCh[0] = nuTotalNumChs;`	
`  if (bEmbeddedSixChFlag){`	
`    nDecCh[nEmDM] = 6;`	
`    nEmDM = nEmDM + 1;`	
`  }`	
`  if (bEmbeddedStereoFlag){`	
`    nDecCh[nEmDM] = 2;`	
`    nEmDM = nEmDM + 1;`	
`  }`	
`  for (ns=0; ns<nuNumMixOutConfigs; ns++){ //Configuration Loop`	
`    for ( nE=0; nE<nEmDM; nE++){ // Embedded downmix loop`	
`      for (nCh=0; nCh<nDecCh[nE]; nCh++){  //Supplemental Channel Loop`	
`        nuMixMapMask[ns][nE][nCh]= ExtractBits(nNumMixOutCh[ns]);`	nNumMixOutCh[ns]
`        nuNumMixCoefs[ns][nE][nCh] =`	
`CountBitsSet_to_1(nuMixMapMask[ns][nE][nCh]);`	
`        for (nC=0; nC<nuNumMixCoefs[ns][nE][nCh]; nC++)`	
`             nuMixCoeffs[ns][nE][nCh][nC] = ExtractBits(6);`	6
`      } // End supplemental channel loop`	
`    }  // End of Embedded downmix loop`	
`  } // End configuration loop`	
`} // End if (bMixMetadataPresent)`	

Table 7-7: Audio Asset Descriptor Syntax: Decoder Navigation Data

Audio asset Descriptor Syntax	Size (Bits)
`nuCodingMode = ExtractBits(2);`	2
`switch (nuCodingMode){`	
`case 0:`	
`   nuCoreExtensionMask = ExtractBits(12);`	12
`   If (nuCoreExtensionMask & DTS_EXSUBSTREAM_CORE)`	
`     nuExSSCoreFsize = ExtractBits(14)+1;`	14
`     bExSSCoreSyncPresent  = ExtractBits(1);`	1
`     if (bExSSCoreSyncPresent)`	
`        nuExSSCoreSyncDistInFrames = 1<<(ExtractBits(2));`	2
`   }`	
`   If (nuCoreExtensionMask & DTS_EXSUBSTREAM_XBR)`	
`     nuExSSXBRFsize = ExtractBits(14)+1;`	14
`   If (nuCoreExtensionMask & DTS_EXSUBSTREAM_XXCH)`	
`     nuExSSXXCHFsize = ExtractBits(14)+1;`	14
`   If (nuCoreExtensionMask & DTS_EXSUBSTREAM_X96)`	
`     nuExSSX96Fsize = ExtractBits(12)+1;`	12
`   If (nuCoreExtensionMask & DTS_EXSUBSTREAM_LBR){`	
`     nuExSSLBRFsize = ExtractBits(14)+1;`	14
`     bExSSLBRSyncPresent = ExtractBits(1);`	1
`     if (bExSSLBRSyncPresent)`	
`        nuExSSLBRSyncDistInFrames = 1<<(ExtractBits(2));`	2
`   }`	
`   If (nuCoreExtensionMask & DTS_EXSUBSTREAM_XLL){`	
`     nuExSSXLLFsize = ExtractBits(nuBits4ExSSFsize)+1;`	nuBits4ExSSFsize
`     bExSSXLLSyncPresent = ExtractBits(1);`	1
`     if (bExSSXLLSyncPresent){`	
`       nuPeakBRCntrlBuffSzkB = ExtractBits(4)<<4;`	4
`       nuBitsInitDecDly  = ExtractBits(5)+1;`	5
`       nuInitLLDecDlyFrames=ExtractBits(nuBitsInitDecDly);`	nuBitsInitDecDly
`       nuExSSXLLSyncOffset=ExtractBits(nuBits4ExSSFsize);`	nuBits4ExSSFsize
`     }`	
`   }`	
`   If (nuCoreExtensionMask & RESERVED_1)`	
`     Ignore = ExtractBits(16);`	16
`   If (nuCoreExtensionMask & RESERVED_2)`	
`     Ignore = ExtractBits(16);`	16
`   Break;`	
`case 1:`	
`   nuExSSXLLFsize = ExtractBits(nuBits4ExSSFsize)+1;`	nuBits4ExSSFsize
`   bExSSXLLSyncPresent = ExtractBits(1);`	1

Audio asset Descriptor Syntax	Size (Bits)
`If (bExSSXLLSyncPresent){`	
`  nuPeakBRCntrlBuffSzkB = ExtractBits(4)<<4;`	4
`  NuBitsInitDecDly  = ExtractBits(5)+1;`	5
`  nuInitLLDecDlyFrames  = ExtractBits(nuBitsInitDecDly );`	nuBitsInitDecDly
`  nuExSSXLLSyncOffset=ExtractBits(nuBits4ExSSFsize);`	nuBits4ExSSFsize
`}`	
`break;`	
`case 2:`	
`nuExSSLBRFsize = ExtractBits(14)+1;`	14
`bExSSLBRSyncPresent = ExtractBits(1);`	1
`  if (bExSSLBRSyncPresent)`	2
`    nuExSSLBRSyncDistInFrames = 1<<(ExtractBits(2));`	
`Break;`	
`case 3:`	
`nuExSSAuxFsize = ExtractBits(14)+1;`	14
`nuAuxCodecID = ExtractBits(8);`	8
`bExSSAuxSyncPresent = ExtractBits(1);`	1
`if (bExSSAuxSyncPresent)`	3
`  nuExSSAuxSyncDistInFrames = ExtractBits(3)+1;`	
`Break;`	
`default:`	
`break;`	
`}`	
`if (  ((nuCodingMode==0) && (nuCoreExtensionMask & DTS_EXSUBSTREAM_XLL))`	
`|| (nuCodingMode==1) ){`	
`nuDTSHDStreamID = ExtractBits(3);`	3
`}`	
`if (bOne2OneMapChannels2Speakers ==true && bMixMetadataEnbl ==true &&`	
`bMixMetadataPresent==false)`	1
`bOnetoOneMixingFlag = ExtractBits(1);`	
`if (bOnetoOneMixingFlag) {`	
`bEnblPerChMainAudioScale = ExtractBits(1);`	1
`for (ns=0; ns<nuNumMixOutConfigs; ns++){`	
`  if (bEnblPerChMainAudioScale){`	
`    for (nCh=0; nCh<nNumMixOutCh[ns]; nCh++)`	
`      nuMainAudioScaleCode[ns][nCh] = ExtractBits(6);`	
`  }`	6
`  else`	
`    nuMainAudioScaleCode[ns][0] = ExtractBits(6);`	
`}`	
`} // End of bOnetoOneMixingFlag==true condition`	
`bDecodeAssetInSecondaryDecoder = ExtractBits(1);`	1
`bDRCMetadataRev2Present = (ExtractBits(1) == 1) ? TRUE : FALSE;`	1
`if (bDRCMetadataRev2Present == TRUE)`	
` {`	
`   DRCversion_Rev2 = ExtractBits(4);`	4
`  // one DRC value for each block of 256 samples`	
`  nRev2_DRCs  = nuExSSFrameDurationCode / 256;`	
`  // assumes DRCversion_Rev2 == 1:`	
`  for (subSubFrame=0; subSubFrame < nRev2_DRCs; subSubFrame++)`	
`  {`	
`    DRCCoeff_Rev2[subSubFrame] = dts_dynrng_to_db(ExtractBits(8));`	8 * nRev2_DRCs
`  }`	
` }`	
`Reserved = ExtractBits(…);`	…
`ZeroPadForFsize = ExtractBits(0 … 7);`	0...7

7.5.3.2 Static Metadata

nuAssetDescriptFsize (Size of Audio Asset Descriptor in Bytes)

This field indicates the size of the audio asset descriptor in bytes, from nuAssetDescriptFsize to ZeroPadForFsize inclusive. If there are multiple audio assets, nuAssetDescriptFsize is used to navigate to the location of the next audio asset descriptor.

nuAssetIndex (Audio Asset Identifier)

This parameter represents the unique audio asset index. Its range is from 0 to nuNumAssets - 1. If the current asset is to be mixed into some lower index asset and the current extension substream index is zero (nExtSSIndex = 0), the current asset index shall be greater than zero (nuAssetIndex > 0), even if it is the first asset to be encoded in the extension substream 0. This condition guarantees correct decoder behaviour when the primary audio does not contain a DTS-HD extension substream. This can be either DTS core substream only or non-DTS primary audio. It is assumed that primary audio is carried as the audio asset 0 in the extension substream 0.

bAssetTypeDescrPresent (Asset Type Descriptor Presence)

This field is present only if bStaticFieldsPresent is true. When set, it indicates that the audio asset type descriptor field follows.

nuAssetTypeDescriptor (Asset Type Descriptor)

This field is present only if bStaticFieldsPresent and bAssetTypeDescrPresent are true. This field represents the index into the audio asset type descriptor lookup table shown in Table 7-8. The value of nuAssetTypeDescriptor = 15 is reserved for future expanded audio asset types. Decoders that do not have an expanded asset type table definition shall treat this case as an "Unknown" audio asset type.

Table 7-8: Audio Asset Type Descriptor Table

nuAssetTypeDescriptor	Comment
0	Music
1	Effects
2	Dialog
3	Commentary
4	Visually impaired
5	Hearing impaired
6	Isolated music object (group of instruments/voices)
7	Music and Effects
8	Dialog and Commentary
9	Effects and Commentary
10	Isolated music object and Commentary
11	Isolated music object and Effects
12	Karaoke
13	Music, Effects and Dialog
14	Complete Audio Presentation
15	RESERVED

bLanguageDescrPresent (Language Descriptor Presence)

This field is present only if bStaticFieldsPresent is true. When set, it indicates that a language descriptor field follows.

LanguageDescriptor (Language Descriptor)

This field is present only if bStaticFieldsPresent and bLanguageDescrPresent are true. It denotes that a language code is associated with the audio asset.

The code represents a 3-character language identifier according to the ISO 639-2 [9]. Each character is coded using 8 bits according to ISO/IEC 8859-1 [10] (ISO Latin-1) and inserted in order into the 24-bit field (1^{st} character = 8 MSBs … 3^{rd} character = 8 LSBs).

bInfoTextPresent (Additional Textual Information Presence)

This field is present only if bStaticFieldsPresent is true. When set, it indicates that additional textual information fields are present.

nuInfoTextByteSize (Byte Size of Additional Text Info)

This field is present only if bStaticFieldsPresent and bInfoTextPresent are true. The value indicates the size in bytes of the text information field.

InfoTextString (Additional Textual Information String)

This field is present only if bStaticFieldsPresent and bInfoTextPresent are true. This character string may be used as a textual description of the audio asset. Each character in the string is encoded with 1-byte using the ISO Latin-1 alphabet (ISO/IEC 8859-1 [10]).

nuBitResolution (PCM Bit Resolution)

This field is present only if bStaticFieldsPresent is true. It expresses the original resolution of the PCM audio source as output by an A/D converter. If the sample resolution differs between the channels, nuBitResolution indicates the maximum value among all channels in the audio asset.

nuMaxSampleRate

This field is present only if bStaticFieldsPresent is true. This field contains the index to the sample frequency, as defined in Table 7-9. It allows the system to quickly ascertain the maximum sample frequency among all channels present in the audio asset.

Table 7-9: Source Sample Rate Table

nuMaxSampleRate	Sample Frequency (Hz)
0	8 000
1	16 000
2	32 000
3	64 000
4	128 000
5	22 050
6	44 100
7	88 200
8	176 400
9	352 800
10	12 000
11	24 000
12	48 000
13	96 000
14	192 000
15	384 000

nuTotalNumChs

This field is present only if bStaticFieldsPresent is true. It represents the total number of channels that might be decoded individually. This field can range from 0 to 255 and the total number of channels can range from 1 to 256.

bOne2OneMapChannels2Speakers

This field is present only if bStaticFieldsPresent is true. If true, this flag indicates that each encoded channel within this audio asset represents a signal feed to a corresponding loudspeaker on the decode side. If this flag is false, it indicates that channels within this audio asset carry the signals that describe the sound field, but are not actual loudspeaker feeds. The actual loudspeaker feeds shall be derived on the decode side using the stream-embedded coefficients and possible user-provided adjustment factors.

bEmbeddedStereoFlag

This is present only when bStaticFieldsPresent and bOne2OneMapChannels2Speakers are true and the total number of channels is greater than two (nuTotalNumChs > 2). When this flag is true, an embedded stereo downmix, generated during encoding, exists in the asset as a separate channel set in the front left and right channel pair (LR). When this is true, a decoder attempting to decode only two channels shall extract and decode the LR pair that already contains the stereo downmix and ignore all other channels. If the bEmbeddedStereoFlag = false, no embedded downmix is generated by the encoder.

bEmbeddedSixChFlag

This is present only when bStaticFieldsPresent and bOne2OneMapChannels2Speakers are true and if the total number of channels is greater than six (nuTotalNumChs > 6). When bEmbeddedSixChFlag = true, this indicates to the decoder that on the encode side, all relevant channels have been mixed:

- into the standard 5.1 layout L, R, C, Ls, Rs, LFE; or
- into the L, R, C, Lss, Rss, LFE layout when the 7.1 layout is in the L, R, C, Lss, Rss, LFE, Lsr and Rsr configuration.

In this case, the decoder that attempts to decode only six channels may extract and decode the 5.1 channel downmix and ignore all other channels. If the bEmbeddedSixChFlag = false, the encoder did not perform the downmix to six channels' operation.

The bEmbeddedSixChFlag field is part of the overview information and may be used by the system for quick access to the format description. All decoders shall extract and use this information during the extraction of the remaining extension substream header fields. After the extension substream header extraction, all decoders shall ignore this information, since more detailed information is available within the channel set sub-headers.

bSpkrMaskEnabled

This is present only when bStaticFieldsPresent and bOne2OneMapChannels2Speakers are true. When bSpkrMaskEnabled is true, all loudspeaker locations are specified using predefined locations as listed in Table 7-10.

nuNumBits4SAMask (Number of Bits for Speaker Activity Mask)

This is present only if bStaticFieldsPresent, bOne2OneMapChannels2Speakers and bSpkrMaskEnabled are true. The value indicates how many bits are used for packing the loudspeaker activity mask nuSpkrActivityMask and the standard speaker layout mask nuStndrSpkrLayoutMask. Valid values for nuNumBits4SAMask are 4, 8, 12 and 16 and are obtained from the extracted 2-bit code using the following mapping:

$$nuNumBits4SAMask = (code + 1) << 2$$

nuSpkrActivityMask (Loudspeaker Activity Mask)

This is present only if bStaticFieldsPresent, bOne2OneMapChannels2Speakers and bSpkrMaskEnabled are true. The nuSpkrActivityMask indicates which of the pre-defined loudspeaker positions apply to the audio channels encoded in DTS-HD stream. Each encoded channel or channel pair, depending on the corresponding speaker position(s), sets the appropriate bit in a loudspeaker activity mask. Predetermined loudspeaker positions are described in Table 7-10. For example, nuSpkrActivityMask = 0xF indicates activity of C, L, R, L_s, R_s and LFE_1 loudspeakers.

When the decoder supports up to a MaxSpeakers number of speakers and when that number is less than the number indicated by the total number of channels, i.e. MaxSpeakers < nuTotalNumChs, the decoder decodes channel sets, in increasing channel set index order, such that the total number of decoded channels is less or equal to MaxSpeakers channels. A channel set with index k, which brings the cumulative number of decoded channels, in all channel sets from 0 to k, to be greater than MaxSpeakers is ignored and not decoded. All channel sets with index >k are also ignored and not decoded. In other words only channel sets with index from 0 to k-1 are decoded.

Table 7-10: Loudspeaker Bit Masks for nuSpkrActivityMask, nuStndrSpkrLayoutMask, nuMixOutChMask

Notation	Loudspeaker Location Description	Bit Mask	Number of Channels
C	Centre in front of listener	0x0001	1
LR	Left/Right in front	0x0002	2
L_sR_s	Left/Right surround on side in rear	0x0004	2
LFE_1	Low frequency effects subwoofer	0x0008	1
C_s	Centre surround in rear	0x0010	1
L_hR_h	Left/Right height in front	0x0020	2
$L_{sr}R_{sr}$	Left/Right surround in rear	0x0040	2
C_h	Centre Height in front	0x0080	1
O_h	Over the listener's head	0x0100	1
L_cR_c	Between left/right and centre in front	0x0200	2
L_wR_w	Left/Right on side in front	0x0400	2
$L_{ss}R_{ss}$	Left/Right surround on side	0x0800	2
LFE_2	Second low frequency effects subwoofer	0x1000	1
$L_{hs}R_{hs}$	Left/Right height on side	0x2000	2
C_{hr}	Centre height in rear	0x4000	1
$L_{hr}R_{hr}$	Left/Right height in rear	0x8000	2

nuNumSpkrRemapSets (Number of Speaker Remapping Sets)

This is present only if bStaticFieldsPresent and bOne2OneMapChannels2Speakers are true. This parameter indicates the number of standard loudspeaker layouts for which remapping coefficients are provided in the stream.

nuStndrSpkrLayoutMask (Standard Loudspeaker Layout Mask)

This is present only if bStaticFieldsPresent and bOne2OneMapChannels2Speakers are true and if the number of speaker remapping sets is greater than zero, (nuNumSpkrRemapSets > 0). nuStndrSpkrLayoutMask indicates the standard speaker layout for which the remapping coefficients are provided. Coding of this field follows the same format as used for **nuSpkrActivityMask**, described in Table 7-10.

nuNumDecCh4Remap (Number of Channels to be Decoded for Speaker Remapping)

This is present only if bStaticFieldsPresent and bOne2OneMapChannels2Speakers are true and if the number of speaker remapping sets is greater than zero (nuNumSpkrRemapSets > 0). For each loudspeaker remapping coefficient set that maps to a particular standard loudspeaker configuration, there are a specific number of encoded channels involved in the remapping. This is the number of channels that needs to be decoded in order to produce the speaker feeds for the standard loudspeaker configurations.

nuRemapDecChMask (Decoded Channels to Output Speaker Mapping Mask)

This is present only if bStaticFieldsPresent and bOne2OneMapChannels2Speakers are true and if the number of speaker remapping sets is greater than zero (nuNumSpkrRemapSets > 0). Usually, a sub-set of all decoded channels is involved in remapping to a particular output loudspeaker and the bit-mask that defines this sub-set is transmitted in the parameter nuRemapDecChMask. Location of bits set to "1" indicates that the corresponding decoded channel is involved in mapping to the particular loudspeaker. The ordering of the channels is first according to the channel set index and next according to the channel mask nuSpkrActivityMask within each channel set. The location of a least significant bit corresponds to the first decoded channel.

nuSpkrRemapCodes (Loudspeaker Remapping Codes)

This is present only if bStaticFieldsPresent and bOne2OneMapChannels2Speakers are true and if the number of speaker remapping sets is greater than zero (nuNumSpkrRemapSets > 0). It indicates the mapping from the encoded channels that are arranged in some non-standard loudspeaker configuration to the specified "standard" loudspeaker configuration. For each output speaker location, there is a sub-set of encoded channels that are contributing to the output speaker feed. This sub-set is described by nuRemapDecChMask and the number of coefficients per each output loudspeaker is equal to the number of bits that are set to "1" in each corresponding nuRemapDecChMask. The transmitted loudspeaker remapping coefficients are obtained from the extracted 5-bit codes according to the procedure that is described in clause C.4. The decoded channels that have their corresponding bits in nuRemapDecChMask set to "0" are not mapped to a specific speaker; in other words, their mapping coefficient is assumed to be $-\infty$ on a log scale or 0 on a linear scale.

nuRepresentationType (Representation Type)

This is present only if bStaticFieldsPresent is true and bOne2OneMapChannels2Speakers is false. It describes the type of representation according to Table 7-11. This information may be useful in some post-processing tasks. The decoder shall export this information to post-processing functions.

Table 7-11: Representation Type

nuRepresentationType	Description
0b000	Audio Asset for Mixing/Replacement
0b001	Not Applicable
0b010	L_t/R_t Encoded for matrix surround decoding implies nuTotalNumChs=2
0b011	L_h/R_h Audio processed for headphone playback implies nuTotalNumChs=2
0b100	Not Applicable
0b101 - 0b111	Reserved

7.5.3.3 Dynamic Metadata

bDRCCoefPresent (Dynamic Range Coefficient Presence Flag)

When bDRCCoefPresent is true, the Dynamic Range Coefficient(s) (DRC) for a current audio asset is present in the stream.

nuDRCCode (Code for Dynamic Range Coefficient)

This field is present only if bDRCCoefPresent is true. Each 8-bit code is an unsigned integer and it indicates a logarithmic gain value in a range from -31,75 dB to 32 dB in steps of 0,25 dB. This format is identical to the DRC coefficients used for the DTS core.

The calculation of the logarithmic gain value (DRC_dB) from the extracted code (DRC_Code) is:

$$DRC\_dB = -32 + (DRC\_Code + 1) \times 0{,}25$$

To perform the dynamic range compression, the decoder multiplies the decoded audio samples by a linear coefficient obtained from the logarithmic gain value (DRC_dB). All channels of an audio asset are scaled by the same value.

nuDRC2ChDmixCode (DRC for Stereo Downmix)

This field is present only if bDRCCoefPresent and bEmbeddedStereoFlag are true. The calculation of the logarithmic gain value (DRC_dB) from the extracted code (nuDRC2ChDmixCode) is as follows:

$$DRC\_dB = -32 + (nuDRC2ChDmixCode + 1) \times 0{,}25$$

Dynamic range compression applied to the two-channel downmix may differ considerably from the dynamic range compression applied to multi-channel mixes. Consequently, when a two-channel downmix has been embedded on the encoder side, a separate dynamic range coefficient is transmitted for the two-channel downmix. The coding of this field is identical to the coding used for the nuDRCCode. When the decoder outputs a two-channel downmix, the decoder shall use the nuDRC2ChDmixCode for the dynamic range compression.

bDialNormPresent (Dialog Normalization Presence Flag)

When bDialNormPresent is true, the dialog normalization parameter for a current audio asset is present in the stream.

nuDialNormCode (Dialog Normalization Code)

This field is present only if the bDialNormPresent is true. nuDialNormCode indicates the dialog normalization gain and is transmitted as a 5-bit unsigned integer in the range from 0 to 31. The dialog normalization gain (DNG), in dB, is specified by the encoder operator and is used to directly scale the decoder output samples in all channels of an audio asset. The dialog normalization gain (DNG) is obtained from the dialog normalization code (nuDialNormCode) simply by inverting the sign, i.e.

$$DNG = -nuDialNormCode$$

The allowed range for DNG is from 0 dB to -31 dB in steps of -1 dB. In case of lossless decoding, the value of DNG = 0 dB shall enable the lossless reconstruction. When nuDialNormCode is updated, the transition from the previous value to the new value is described in clause C.7.

bMixMetadataPresent (Mixing Metadata Presence Flag)

This field is present only if bMixMetadataEnbl is true. When true, the mixing metadata for this audio asset is present in the stream. When bMixMetadataPresent is false, there is no mixing metadata after the bMixMetadataPresent field and the metadata parameters maintain their values from the previous frame. This field allows the rate of metadata transmission to be controlled. When bMixMetadataEnbl is false, the value for bMixMetadataPresent shall be false.

bExternalMixFlag (External Mixing Flag)

This is present only if bMixMetadataPresent is true. Secondary audio assets in some applications may be exported for mixing outside of the DTS-HD decoder. This field indicates whether the asset is exported (flag is true) or the asset is mixed within the DTS-HD decoder (flag is false). The decoder uses this flag to determine which assets shall be exported.

nuPostMixGainAdjCode (Post Mixing/Replacement Gain Adjustment)

This field is present only if bMixMetadataPresent is true. When mixing multiple assets, the overall mixture is additionally scaled by the post-mix adjustment coefficient obtained from the nuPostMixGainAdjCode that is transmitted in the highest-indexed active asset of the desired audio presentation. All nuPostMixGainAdjCode values that are transmitted in lower-indexed active assets of the desired audio presentation are extracted and ignored. This scaling factor is applied to all speaker outputs after combining the last audio asset with the mixture of all lower-indexed active assets. The scale factor is in the range between -14,5 dB and +15 dB in steps of 0,5 dB. It is coded as a 6-bit unsigned index with a valid range from 1 to 60. The interpretation of the index and the calculation of the linear scale factor are described in clause C.5. When nuPostMixGainAdjCode is updated, the transition from the previous value to the new value is described in clause C.7.

nuControlMixerDRC (Dynamic Range Compression Prior to Mixing)

This is present only if bMixMetadataPresent is true. If bMixMetadataPresent=false the nuControlMixerDRC shall be set to its default value of 2.

This is a 2-bit field that represents an index described in Table 7-12.

Table 7-12: Dynamic Range Compression Prior to Mixing

nuControlMixerDRC	Description
0	Prior to mixing, perform dynamic range compression on the mixture of all lower indexed active assets, using corresponding DRC code limited by the value indicated by nuLimit4EmbeddedDRC
1	Prior to mixing, perform dynamic range compression on the current audio asset only using associated DRC code limited by the value indicated by nuLimit4EmbeddedDRC
2	Prior to mixing, perform dynamic range compression on both the mixture of all lower indexed active assets and the current audio asset, each using its own DRC code limited by the value indicated by nuLimit4EmbeddedDRC
3	Prior to mixing, perform dynamic range compression on both the mixture of all lower indexed active assets and the current audio asset, each using the DRC code transmitted in the field nuCustomDRCCode

nuLimit4EmbeddedDRC (Limit for Mixing Dynamic Range Compression)

This field is present only if bMixMetadataPresent is true and nuControlMixerDRC < 3. It is coded as a 3-bit value representing an index in Table 7-13. If parameter nuLimit4EmbeddedDRC is not transmitted in a stream, its value shall be set to 7.

Table 7-13: Limit for Dynamic Range Compression Prior to Mixing

nuLimit4EmbeddedDRC	Limit for DRC code in % (ScaleDRC × 100)
0	0 (disable prior to mixing DRC)
1	15
2	30
3	45
4	60
5	75
6	90
7	100

By specifying a value for nuLimit4EmbeddedDRC a content provider may limit the dynamic range compression that is applied to audio assets prior to the mixing. In particular a listener may specify the desired amount (DesDRC) of dynamic range compression in the range of 0 % to 100 %. The value (ScaleDRC × 100) coded in the parameter nuLimit4EmbeddedDRC represents the upper limit for the original listener requested amount of dynamic range compression (DesDRC). Therefore the decoder shall adjust its logarithmic gain value (DRC_dB corresponding to a DRCCode extracted from a stream) as follows:

```
if (DesDRC>ScaleDRC*100)
    DRC_dB = DRC_dB*ScaleDRC
else
    DRC_dB = DRC_dB*(DesDRC/100).
```

nuCustomDRCCode (Custom Code for Mixing Dynamic Range Coefficient)

This field is present only if bDRCCoefPresent is true and nuControlMixerDRC = 3. When mixing audio assets, if custom dynamic range compression is required, an encoder operator may provide a custom DRC code that is transmitted in nuCustomDRCCode field.

Each code is an 8-bit unsigned integer and indicates a logarithmic gain value in a range from -31,75 dB to 32 dB in steps of 0,25 dB.

The calculation of the logarithmic gain value (DRC_dB) from the extracted code (nuCustomDRCCode) is as follows:

$$DRC\_dB = -32 + (nuCustomDRCCode + 1) \times 0{,}25$$

To perform dynamic range compression, the decoder multiplies the decoded audio samples by a linear coefficient obtained from the logarithmic gain value (DRC_dB). All channels of the mixture of all lower indexed active assets and the current audio asset are scaled by the same value.

bEnblPerChMainAudioScale (Scaling Type for Channels of Main Audio)

This field is present in one of the two cases. if:

1) bMixMetadataPresent is true; or

2) bOnetoOneMixingFlag is true.

The above two cases are mutually exclusive since bMixMetadataPresent=true implies that bOnetoOneMixingFlag=false and also bOnetoOneMixingFlag=true implies that bMixMetadataPresent=false.

If bEnblPerChMainAudioScale is false, a single scale factor is present for all main audio channels that are being mixed with the channels of this audio asset. If bEnblPerChMainAudioScale is true, each main audio channel that is being mixed with the channels of the current audio asset has its own scale factor.

nuMainAudioScaleCode (Scaling Parameters of Main Audio)

This is present only if bMixMetadataPresent is true or if bOnetoOneMixingFlag is true for each mixing configuration independently.

It modifies the gain of the audio program that is being mixed with the current audio program. The number of main audio scale factors is equal to the number of mixer output channels that are defined for a particular mixing configuration. This number is derived by looking at Table 7-10, cumulatively adding the entries in column denoted by "Number of Channels" for each set ("1") bit of nuMixOutChMask. The ordering of the coefficients is derived by checking the bits that are set in the nuMixOutChMask, starting from the LSB and by looking into Table 7-10 for the corresponding channel labels. In case of the channel label that indicates a channel pair, the coefficient for the left channel of the pair comes first. For example the nuSpkrActivityMask = 0x2F indicates a 7.1 ($C+LR+L_sR_s+LFE1+L_hR_h$) mixer configuration and consequently the main audio scale factors for this configuration will be ordered as C, L, R, L_s, R_s, LFE_1, L_h and R_h. The scale factors are obtained from the extracted 6-bit codes (nuMainAudioScaleCode) according to the procedure described in clause C.6. When nuMainAudioScaleCode is updated, the transition from the previous value to the new value is described in clause C.7.

nuMixMapMask (Mix Output Mask)

This is present only if bMixMetadataPresent is true, for each of the output mixing configurations, for each embedded downmix configuration present in the current audio asset and for each channel of a specific embedded downmix configuration. This mask defines the mixing map from each channel of current audio asset to output mixer. Note that, in this context, the full mix is categorized as one of the embedded downmix configurations with the number of channels equal to the nuTotalNumChs.

The number of mixer output channels that are defined for a particular mixing configuration is derived from the nuMixOutChMask. Each channel of the current audio asset may be mapped to any of the mixer output channels using the mixing coefficients described in nuMixCoeffs.

The nuMixMapMask has a dedicated bit (channel flag) for each nNumMixOutCh mixer output channel, keeping them in the same order as defined by the nuMixOutChMask. In particular the order of channel flags in the nuMixMapMask, when starting from the LSB, is derived by checking the bits that are set in the nuMixOutChMask starting from the LSB and by looking into Table 7-10 for the corresponding channel labels. In case of the channel label that indicates a channel pair, the channel flag for the left channel of the pair comes first.

A channel of the current audio asset is mapped exclusively to those mixer output channels that have their bits in nuMixMapMask set to "1". Mapping to all other output channels assumes mixing coefficient equal to $-\infty$ dB.

nuMixCoeffs (Mixing Coefficients)

This is present only if bMixMetadataPresent is true, for each of the output mixing configurations, each embedded downmix configuration present in the current audio asset and each channel of a specific embedded downmix configuration. Note that, in this context, the full mix is categorized as one of the embedded downmix configurations, with the number of channels equal to the nuTotalNumChs.

The scale factors are obtained from the extracted 6-bit codes according to the procedure that is described in clause C.6. The transition from the previous value to the new value when nuMixCoeffs is updated is described in clause C.7.

7.5.3.4 Decoder Navigation Data

nuCodingMode (Coding Mode for the Asset)

This is a 2-bit field that represents an index into a look-up table. The look-up table, Table 7-14, describes the audio coding modes that may be used for compression of audio in the asset.

Table 7-14: Coding Mode

nuCodingMode	Description
0	DTS-HD Coding Mode that may contain multiple coding components
1	DTS-HD Loss-less coding mode without CBR component
2	DTS-HD Low bit-rate mode
3	Auxiliary coding mode

The auxiliary coding mode is reserved for future applications.

nuCoreExtensionMask (Coding Components Used in Asset)

This field is present only when nuCodingMode = 0. This is an array of 12 bits indicating the core and the extensions that are used for coding the current audio asset. A '1' in the mask at the bit location reserved for a specific coding component indicates that the particular coding component is used in this audio asset. A '0' in the mask at the bit location reserved for a specific coding component indicates that the particular coding component is NOT used in this audio asset. The association of bit locations within nuCoreExtensionMask to the specific coding components is described in, Table 7-15, shown below. Data for each substream type within an audio asset will appear in the same order (LSB first) as the bit mask position.

Table 7-15: Core/Extension Mask

Notation	Core/Extension Type Description	CBR or VBR	nuCoreExtensionMask
DTS_CORESUB_STREAM_CORE	Core component within the core substream	CBR	0x001
DTS_BCCORE_XXCH	XXCH extension, when combined with associated core, may be sent via SPDIF for backward compatibility	CBR	0x002
DTS_ BCCORE _X96	X96 extension, when combined with associated core, may be sent via SPDIF for backward compatibility	CBR	0x004
DTS_ BCCORE_XCH	XCH extension, when combined with associated core, may be sent via SPDIF for backward compatibility	CBR	0x008
DTS_EXSUB_STREAM_CORE	Core component within the current extension substream	CBR	0x010
DTS_EXSUB_STREAM_XBR	XBR extension within the current extension substream	CBR	0x020
DTS_EXSUB_STREAM_XXCH	XXCH extension within the current extension substream	CBR	0x040
DTS_EXSUB_STREAM_X96	X96 extension within the current extension substream	CBR	0x080
DTS_EXSUB_STREAM_LBR	Low bit rate component within the current extension substream	CBR	0x100
DTS_EXSUB_STREAM_XLL	Lossless extension within the current extension substream	VBR	0x200
RESERVED_1			0x400
RESERVED_2			0x800

The bit locations 0x400 and 0x800 within the nuCoreExtensionMask are reserved for future use and all the decoders that comply with this version of the specification shall ignore all coding components that have any of these 2 bits set to "1".

nuExSSCoreFsize (Size of Core Component in Extension Substream)

This field is present only when nuCodingMode = 0 and the DTS_EXSUB_STREAM_CORE bit of nuCoreExtensionMask is set to "1".

This field indicates per frame payload size, in bytes, of a core component of the audio asset present in the extension substream. When the asset consists of core and extension(s) that are carried in the extension substream, the nuExSSCoreFsize is used to navigate to the location of a first extension as indicated by the nuCoreExtensionMask.

bExSSCoreSyncPresent (Core Sync Word Present Flag)

This field is present only when nuCodingMode = 0 and the DTS_EXSUB_STREAM_CORE bit of nuCoreExtensionMask is set to "1".

If bExSSCoreSyncPresent is true, the sync word for the associated core component is present. This indicates to the asset decoder that it should attempt to establish/verify synchronization in the current frame.

If bExSSCoreSyncPresent is false, no core sync word is present in the current frame of extension substream. If the asset decoder is not synchronized, it is not able to establish synchronization. If the asset decoder is synchronized, it consumes the nuExSSCoreFsize bytes of the core data.

nuExSSCoreSyncDistInFrames (Core Sync Distance)

This 2-bit field is present only if bExSSCoreSyncPresent is true. The nuExSSCoreSyncDistInFrames represents the distance between the two sync words in the core stream of the current asset measured in number of extension substream frames. The nuExSSCoreSyncDistInFrames takes values 1, 2, 4 and 8, which are obtained from the transmitted 2-bit codes 0, 1, 2 and 3 respectively.

nuExSSXBRFsize (Size of XBR Extension in Extension Substream)

This field is present only when nuCodingMode = 0 and the DTS_EXSUB_STREAM_XBR bit of nuCoreExtensionMask is set to "1".

This field indicates per frame payload size, in bytes, of XBR extension of the audio asset present in the extension substream. When the asset contains additional extensions that follow the XBR extension in the extension substream, the nuExSSXBRFsize is used to navigate to the location of the next extension as indicated by the nuCoreExtensionMask.

nuExSSXXChFsize (Size of XXCH Extension in Extension Substream)

This field is present only when nuCodingMode = 0 and DTS_EXSUB_STREAM_XXCH in nuCoreExtensionMask is set to "1".

This field indicates per frame payload size, in bytes, of XXCH extension of the audio asset present in the extension substream. When the asset contains additional extensions that follow the XXCH extension in the extension substream, the nuExSSXXCHFsize is used to navigate to the location of the next extension as indicated by the nuCoreExtensionMask.

nuExSSX96Fsize (Size of X96 Extension in Extension Substream)

This field is present only when nuCodingMode = 0 and the DTS_EXSUB_STREAM_X96 bit of nuCoreExtensionMask is set to "1".

This field indicates per frame payload size, in bytes, of X96 extension of the audio asset present in the extension substream. When the asset contains additional extensions that follow the X96 extension in the extension substream, the nuExSSX96Fsize is used to navigate to the location of the next extension as indicated by the nuCoreExtensionMask.

nuExSSLBRFsize (Size of LBR Component in Extension Substream)

This field is present when:

- nuCodingMode = 0 and DTS_EXSUB_STREAM_LBR in nuCoreExtensionMask is set to "1"; or
- nuCodingMode = 2, which indicates that the asset is coded using only a low bit rate component.

This field indicates per frame payload size, in bytes, of LBR component of the audio asset present in the extension substream. When the asset contains additional extensions that follow the LBR component in the extension substream, the nuExSSLBRFsize is used to navigate to the location of the next extension as indicated by the nuCoreExtensionMask.

bExSSLBRSyncPresent (LBR Sync Word Present Flag)

This field is present when:

- nuCodingMode = 0 and DTS_EXSUB_STREAM_LBR in nuCoreExtensionMask is set to "1"; or
- nuCodingMode = 2, indicating the asset is coded using only a low bit rate component.

If bExSSLBRSyncPresent is true for an asset in the current frame of the extension substream, the sync word for the associated LBR component is present. This indicates to the asset decoder that it should attempt to establish/verify synchronization in the current frame.

If bExSSLBRSyncPresent is false for an asset, no LBR sync word is present in the current frame of extension substream and the asset decoder is not capable, when starting from unsynchronized state, to establish the synchronization with the LBR data present in this frame. If the asset decoder is in a synchronized state, it consumes the nuExSSLBRFsize bytes of the LBR data.

nuExSSLBRSyncDistInFrames (LBR Sync Distance)

This 2-bit field is present only if bExSSLBRSyncPresent is true. The nuExSSLBRSyncDistInFrames represents the distance between the two sync words in the LBR stream of the current asset measured in number of extension substream frames. The nuExSSLBRSyncDistInFrames takes values 1, 2, 4 and 8, which are obtained from the transmitted 2-bit codes 0, 1, 2 and 3 respectively.

nuExSSXLLFsize (Size of XLL Data in Extension Substream)

This field is present when either:

- nuCodingMode = 0 and DTS_EXSUB_STREAM_XLL in nuCoreExtensionMask is set to "1"; or
- nuCodingMode = 1, which indicates that asset is coded using only a lossless component.

This field indicates the size, in bytes, of XLL extension of the audio asset present in the extension substream.

bExSSXLLSyncPresent (XLL Sync Word Present Flag)

This field is present when either:

- nuCodingMode = 0 and DTS_EXSUB_STREAM_XLL in nuCoreExtensionMask is set to "1"; or
- nuCodingMode = 1, which indicates that asset is coded using only a lossless component.

If the bExSSXLLSyncPresent is true for an asset in current frame of the extension substream, the sync word for the associated XLL component is present. This indicates to the asset decoder that it should attempt to establish/verify synchronization in the current frame.

If bExSSXLLSyncPresent is false for an asset, no XLL sync word is present in the current frame of extension substream and the asset decoder is not capable, when starting from unsynchronized state, to establish the synchronization with the XLL data present in this frame. If the asset decoder is in a synchronized state it consumes the nuExSSXLLFsize bytes of the XLL data.

nuPeakBRCntrlBuffSzkB (Peak bit rate smoothing buffer size)

This field is present when bExSSXLLSyncPresent is true and when either:

- nuCodingMode = 0 and DTS_EXSUB_STREAM_XLL in nuCoreExtensionMask is set to "1"; or
- nuCodingMode = 1.

This field represents the size, in kBytes, of the peak bit rate smoothing buffer, which has been assumed to exist on the decode side during a lossless encoding/authoring of the current asset.

The available sizes are from 0 Kbyte to 240 kBytes in steps of 16 kBytes.

Each lossless encoded asset has its corresponding smoothing buffer and nominally the sum of buffer sizes for all active assets shall not exceed the specified decoder's buffer size. However, next-generation streams may be created such that for a certain subset of all active assets, all decoders meet the buffer size requirements, but for decoding of all active assets, the new generation decoders are required. Decoders capable of decoding only a subset of all active assets (because of limitation in available buffer size) shall ignore all other assets that do not belong to the specified subset. This subset consists of the lowest index active assets that jointly do not require more than available decoder buffer size.

nuBitsInitDecDly (Size of field nuInitLLDecDlyFrames)

This field is present when bExSSXLLSyncPresent is true and when either:

- nuCodingMode = 0 and DTS_EXSUB_STREAM_XLL in nuCoreExtensionMask is set to "1"; or
- nuCodingMode = 1.

The nuBitsInitDecDly is the number of bits used to extract the parameter nuInitLLDecDlyFrames (Initial XLL Decoding Delay in Frames).

nuInitLLDecDlyFrames (Initial XLL Decoding Delay in Frames)

This field is present when bExSSXLLSyncPresent is true and when either:

- nuCodingMode = 0 and DTS_EXSUB_STREAM_XLL in nuCoreExtensionMask is set to "1"; or
- nuCodingMode = 1.

The nuInitLLDecDlyFrames is the number of frames to delay lossless decoding of the current asset after the initial synchronization is established. It instructs the decoder to wait 'nuInitLLDecDlyFrames' frames before decoding the first frame of the current asset, after establishing/re-establishing the synchronization. For all consecutive frames, as long as the decoder is synchronized, the decoder ignores the nuInitLLDecDlyFrames.

This value tells the decoder how many frames the lossless frame in question needs to be delayed until it can be decoded. The frame is placed into the decoder's buffer and interprets the offset as a time stamp of the delay. The decoder receives the data, detects the offset number and does not decode the frame until the specified offset number of frames intervals has elapsed.

The frame offset specifies a schedule of when the data is in the buffer and when it is to be decoded. If the stream consists of both lossy and lossless substreams the decoder shall decode lossy data and output the lossy decoded audio while waiting for 'nuInitLLDecDlyFrames' delay to expire. If the stream consists of lossless data only, decoder outputs shall be muted until the 'nuInitLLDecDlyFrames' delay expires.

nuExSSXLLSyncOffset (Number of Bytes Offset to XLL Sync)

This field is present when bExSSXLLSyncPresent is true and when either:

- nuCodingMode = 0 and DTS_EXSUB_STREAM_XLL in nuCoreExtensionMask is set to "1"; or
- nuCodingMode = 1.

This specifies the number of bytes offset (from start of XLL data in current asset) to locate the first XLL sync word in the current asset.

nuExSSAuxFsize (Size of Auxiliary Coded Data)

This field is present only when nuCodingMode = 3.

This field indicates the size, in bytes, of auxiliary coded data present in the extension substream for the current audio asset.

nuAuxCodecID (Auxiliary Codec Identification)

This 8-bit field is present only if nuCodingMode = 3. Its value represents an index into the auxiliary codec lookup table. This feature is not supported in the current version of DTS-HD.

bExSSAuxSyncPresent (Aux Sync Word Present Flag)

This field is present only when nuCodingMode = 3.

If the bExSSAuxSyncPresent is true for an asset in the current frame of the extension substream, the sync word for the associated auxiliary component is present. This indicates to the asset decoder that it should attempt to establish/verify synchronization in the current frame.

If bExSSAuxSyncPresent is false for an asset, no auxiliary sync word is present in the current frame of extension substream and the asset decoder is not capable, when starting from unsynchronized state, to establish the synchronization with the auxiliary coded data present in this frame. If the asset decoder is in a synchronized state, it consumes the nuExSSAuxFsize bytes of the auxiliary coded data.

nuExSSAuxSyncDistInFrames (Aux Sync Distance)

This 3-bit field is present only if bExSSAuxSyncPresent is true. The nuExSSAuxSyncDistInFrames represents the distance between the two sync words in the auxiliary stream of the current asset measured in number of extension substream frames. The nuExSSAuxSyncDistInFrames takes values in a range from 1 to 8.

nuDTSHDStreamID (DTS-HD Stream ID)

This 3-bit field is present only if the XLL coding component is present in the asset. It represents the unique ID number (ranging from 0 to 7) of a DTS-HD stream that carries the asset data. All assets within the same DTS-HD stream would have the same value for the nuDTSHDStreamID. This parameter is used to indicate to the decoder that seamless switching between the two DTS-HD streams, both carrying an asset with the XLL coding components, has occurred in the player.

If the value of nuDTSHDStreamID changes from the value present in the previous frame, the XLL stream decoding is disabled in the current (transition) frame and the XLL receive and output buffers are zeroed out.

When the lossy component is used together with the XLL extension, the decoding of the lossy stream is uninterrupted. The audio decoded from the lossy component of the new stream will be played out during the transition frame. The XLL decoding will continue in the following frame according to the buffering delay obtained from the new stream. After the expiration of imposed delay on the XLL decoding, both the lossy and the XLL components are decoded and combined together to generate lossless decoded audio.

When the XLL is used in standalone mode (without lossy component), the audio will be muted in the transition frame and the audio in the frame after the transition frame will be faded in.

bOnetoOneMixingFlag

This flag is present only when bOne2OneMapChannels2Speakers and bMixMetadataEnbl are true and bMixMetadataPresent is false.

This flag when true indicates a simplistic asset mixing scenario in which the current audio asset channels are directly added to the corresponding audio asset, with appropriate scaling of the primary audio channels or appropriate scaling of the current audio asset channels depending upon the bDecodeAssetInSecondaryDecoder flag.

When bOnetoOneMixingFlag is true and bDecodeAssetInSecondaryDecoder is true, the simplistic asset mixing scenario has the current audio asset channels directly added to the corresponding primary audio channels with appropriate scaling of the primary audio channels. In the case of a 5.1 primary audio asset and a 2.0 secondary audio asset, the left secondary audio channel will only be added to the left primary audio channel. Similarly, the right secondary audio channel will only be added to the right primary audio channel. Primary audio channels may be individually scaled prior to the addition of secondary audio channels using the coefficients that are obtained from the codes (nuMainAudioScaleCode) transmitted immediately after the bOnetoOneMixingFlag.

When bOnetoOneMixingFlag is true and bDecodeAssetInSecondaryDecoder is false, the simplistic asset mixing scenario has the current audio asset channels directly added to corresponding audio channels of all active assets with appropriate scaling of the current audio channels. In the case of active audio assets with 5.1 channels, the left audio channel of the current asset will only be added to the left audio channel of other active assets. Similarly, the right audio channel of the current asset will only be added to the right audio channel of other active assets. The current audio channels may be individually scaled prior to the addition with other active asset audio channels using the coefficients that are obtained from the codes (nuMainAudioScaleCode) transmitted immediately after the bOnetoOneMixingFlag.

The mixing scenario described above minimizes the amount of mixing metadata that needs to be transmitted.

bDecodeAssetInSecondaryDecoder

This flag when true indicates that the current asset is to be decoded by the secondary decoder. In particular the primary decoder shall be able to decode at least one active audio asset (with bDecodeAssetInSecondaryDecoder =false) and the secondary decoder decodes up to one active audio asset. This is a guarantee that the solutions implementing both primary and secondary decoders will at least be able to decode two audio assets. It is the responsibility of the external post-process to combine the audio obtained from the primary and secondary decoder. This external process is either controlled by the external metadata or by the metadata present in the audio asset with bDecodeAssetInSecondaryDecoder = true and bExternalMixFlag = true. In later case the external process will request the metadata from the secondary decoder.

bDRCMetadataRev2Present

When the bDRCMetaDataRev2Present flag is TRUE, the Rev2 DRC metadata that enables dynamic range control with a time resolution of one value for every period of 256*RefClockPeriod seconds, will be present. This Rev2 DRC metadata shall be used for controlling the dynamic range of all decoded channels in the current asset instead of:

- the nuDRCCode and the nuDRC2ChDmixCode metadata that may be present in the current asset descriptor; or
- the subsubFrameDRC_Rev2AUX that may be present in a DTS core associated with the current asset.

If the bDRCMetaDataRev2Present flag is FALSE, no Rev2 DRC metadata is present in the current asset descriptor.

DRCversion_Rev2

This field will be present only if bDRCMetaDataRev2Present flag is TRUE. The DRCversion_Rev2 is a four bit field which is used to determine the version of DRC algorithm which the encoder used. The first version starts at 0x1. Decoders will support DRC version 0x1 up to the latest version which they support. Currently only DRCversion_Rev2 = 1 is supported. If the encoder is supplying DRC information with a version number higher than that which is supported by the decoder, the supplied Rev2 DRC values should be ignored and no DRC should be applied.

DRCCoeff_Rev2

This field will be present only if bDRCMetaDataRev2Present flag is TRUE. Currently only DRC version 1 is supported, which is single band mode. In single band mode, one 8 bit value is transmitted every 256*RefClockPeriod seconds. Consequently in a single band mode in each frame there will be nuExSSFrameDurationCode/256 values for DRCCoeff_Rev2.

Each 8-bit value for DRCCoeff_Rev2 is extracted from the bitstream and converted into a dB gain by function dts_dynrng_to_db().

Reserved (Reserved)

This field is reserved for extending the information present in the audio asset descriptor. The decoder shall assume that this field is present and of unspecified duration. Therefore, in order to continue unpacking the stream, the decoder shall skip over this field using the extension substream header start pointer and the audio asset descriptor size nuAssetDescriptFsize.

ZeroPadForFsize (Make nuAssetDescriptFsize a multiple of 8 bits)

This field ensures that the size of an audio asset descriptor is an integer number of bytes. Encoder appends '0's until the current bit position relative to the first packed bit of nuAssetDescriptFsize is a multiple of 8 bits.

8 DTS Lossless Extension (XLL)

8.1 General Information About the XLL Extension

The DTS lossless coding extension (XLL) results in a bit for bit accurate reproduction of the input signal. The lossless encoding system has provisions to support a maximum audio sample rate of 384 kHz and the capability to support high channel counts when used in the DTS-HD framework, which also permits nearly arbitrary speaker mapping.

Since this is a bit accurate system, XLL may be used alone, or it may be used as a residual coder in conjunction with a lossy coding system, as in the current commercial application of DTS-HD Master Audio™. This clause describes the XLL extension.

8.2 Lossless Frame Structure

8.2.1 General Information About the Lossless Frame Structure

As depicted in Figure 8-1, an XLL lossless frame consists of a header and one or more frequency bands. Furthermore, each frequency band consists of one or more data segments. The number of data segments in each frequency band is the same. Time duration of each segment is the same for all segments in the frame.

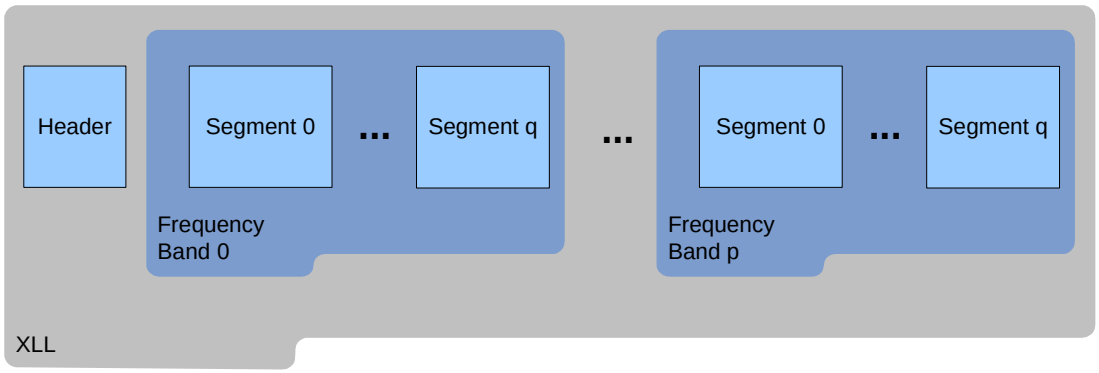


Figure 8-1: XLL Lossless Frame

Both the header and the segments are each further sub-divided by the *channel set* information. The channel set information is a representation of the source material in either downmixed or unmixed form. Similar to the segments, each expresses the same duration in seconds, but not necessarily the same number of samples, because each channel set can represent different sample rates.

By way of example, in a given segment the first set ($ChSET_0$) could represent 5.1 downmixed primary channels with the second set ($ChSET_1$) representing two extra ES channels for reversing the downmix to define a 7.1 discrete environment.

8.2.2 Header Structure

8.2.2.1 General Information About the Header Structure

As shown in Figure 8-2, the lossless frame header consists of a sync word, common setup data (such as the number of frequency bands, the number of segments in the frame, the number of samples in each segment), information specific to each of the channel sets (such as number of channels, channel layout and sampling frequency) and the navigation indices. For example, a frame containing two channel sets consisting of a primary 5.1 set and an ES channel set would be specified by data consisting of the common header field plus two specific channel set sub-headers and a navigation index (NAVI) table.

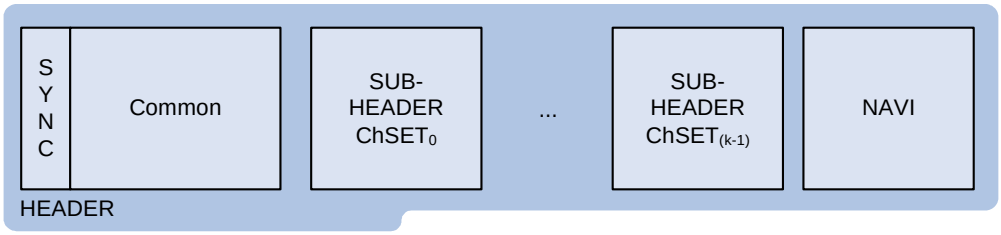


Figure 8-2: XLL Header Structure

8.2.2.2 Common Header

The common header area indicates the header size, which helps locate the first channel set sub-header and the checksum field. It also contains flags indicating if a core is present, if the core should be interpolated, the total size of the entire frame and a frame offset designating the number of frames by which to delay the decoding in order to facilitate buffer management at the decoder. The common header also includes the number of segments per frame and the number of samples in each segment of channel set 1. (All other channel sets scale the number of samples in a segment using the ratio of their sampling frequency and the sampling frequency of the first channel set).

A checksum is included at the end of the common header as a verification measure and to help detect erroneous sync word caused by alias occurrences of the sync word. Since aliases of the sync word are a concern, the decoder shall always calculate the header checksum fields to verify the validity of the detected sync word. To this end, when a valid checksum pattern is discovered, decoding can begin immediately. See Annex B for a complete discussion on Cycle Redundancy Checking (CRC).

8.2.3 Channel Set Sub-Header

Specific header data for each channel set is stored within the channel set sub-header. The channel set sub-header field includes its size, the number of channels in the set, the sampling rate and specific data, such as the replacement flag and downmixing coefficients.

The first channel set sub-header is read by indexing its start position, which is reached by computing the offset frame start position plus the size of the common header. From this position the sub-header data for the first channel set is unpacked. The next channel set sub-header is located by extracting the `mChSetHeaderFSize` of the current channel set and advancing by `mChSetHeaderFSize` bytes from the start of the current channel set.

8.2.4 Navigation Index

The NAVI table consists of the sizes of individual frequency bands. It is described in more detail in clause 8.4.2.

8.2.5 Frequency Band Structure

The frequency band 0 is considered to be a base band and it is always present in the stream. Up to three extended frequency bands may also be present in the stream, supporting the sampling frequency of up to four times the sampling frequency of the base band. For example, all channel sets that have the sampling frequency less than or equal to 96 kHz are coded entirely within the frequency band 0. The channel sets that have a sampling frequency of 192 kHz are coded using both the frequency band 0 and the frequency band 1. Finally, the channel sets that have a sampling frequency of 384 kHz are coded using all frequency bands: 0, 1, 2 and 3.

The number of extended frequency bands is part of the header information. After adding the number of extended bands, the total number of frequency bands can be 1, 2, or 4. On the encode side, the number of frequency bands is determined simply by the underlying maximum sampling frequency among all of the channel sets. In particular, for sampling frequency $F_{s,}$ the number of encoded frequency bands is determined as follows:

- Number of frequency bands is 1 for $F_s \leq Base\_F_s$
- Number of frequency bands is 2 for $Base\_F_s < F_s \leq 2 \times Base\_F_s$
- Number of frequency bands is 4 for $2 \times Base\_Fs < Fs \leq 4 \times Base\_Fs$

where $Base\_F_s$ denotes the base sampling frequency i.e. 64 kHz, 88.2 kHz, or 96 kHz.

A decoder that supports only sampling frequencies up to the $Base\_F_s$ shall decode only the frequency band 0 and skip over the remaining bands. Similarly, a decoder that supports only sampling frequencies up to the $2 \times Base\_F_{s,}$ shall decode only the frequency bands 0 and 1 and skip over the remaining bands.

8.2.6 Segments and Channel Sets

The segments and channel sets further subdivide the frequency band data. A segment within a frequency band contains the encoded frequency band samples, over a specific period of time (segment duration), for all channel sets present in the frequency band.

As shown in Figure 8-3, the encoded and packed channel set data are interleaved into a segment. There are no headers associated with these two levels of abstraction. A segment represents the same time duration for all channels sets when unpacked.

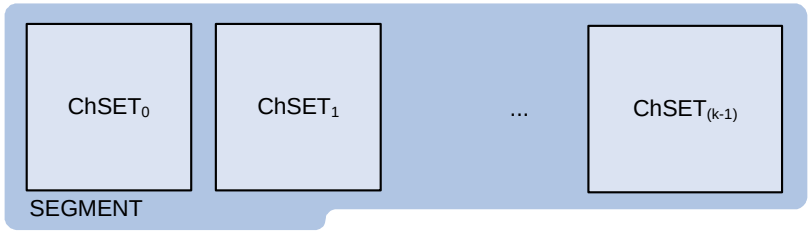


Figure 8-3: Segment with Encoded and Packed Channel Sets

8.3 Lossless Stream Syntax

8.3.1 Common Header

Table 8-1 describes the composition of the common header.

Table 8-1: Common Header

Syntax	Size (Bits)
`SYNCXLL = ExtractBits(32);`	32
`nVersion = ExtractBits(4) + 1;`	4
`nHeaderSize = ExtractBits(8) + 1;`	8
`nBits4FrameFsize = ExtractBits(5) + 1`	5
`nLLFrameSize = ExtractBits(nBits4FrameFsize) + 1;`	nBits4FrameFsize
`nNumChSetsInFrame = ExtractBits(4) + 1;`	4
`tmp = ExtractBits(4);` `nSegmentsInFrame = 1 << tmp;`	4
`tmp = ExtractBits(4);` `nSmplInSeg = 1 << tmp;`	4
`nBits4SSize = ExtractBits(5) + 1;`	5
`nBandDataCRCEn = ExtractBits(2);`	2
`bScalableLSBs = ExtractBits(1);`	1
`nBits4ChMask = ExtractBits(5) + 1;`	5
`if (bScalableLSBs)` `  nuFixedLSBWidth = ExtractBits(4);`	4
`Reserved = ExtractBits(…);`	…
`ByteAlign = ExtractBits(0 … 7)`	0...7
`nCRC16Header = ExtractBits(16);`	16

SYNCXLL (XLL extension sync word)

The lossless DWORD aligned synchronization word has value 0x41A29547. During sync detection the nCRC16Header checksum is used to further verify that the detected sync pattern is not a random alias. The DWORD alignment makes it necessary to append a total of from 1 to 3 extra zero bytes after the last band data of the previous frame.

nVersion (Version number)

This is the lossless stream syntax version identification. If the version indicated here is greater than the version of the decoder, the decoder should not attempt to process the stream further.

nHeaderSize (Lossless frame header length)

This is the size of the common header in bytes from the SYNCXLL to nCRC16Header inclusive. This value determines the location of the first channel set header. This marker also designates the end of the field nCRC16Header and allows quick location of the checksum at byte position nHeaderSize-2.

nBits4FrameFsize (Size of field nBytesFrameLL)

This is the number of bits less one used to store the lossless frame size parameter nLLFrameSize.

nLLFrameSize (Number of bytes in a lossless frame)

This is the total number of bytes in a lossless frame. This value is used to traverse to the end of the frame.

nNumChSetsInFrame (Number of Channel Sets per Frame)

This is the number of channel sets. It is used to loop through the channel set sub-headers, the channel sets in each segment and the NAVI index.

nSegmentsInFrame (Number of Segments per Frame)

This is the number of segments in the current frame. Representing a binary exponent, the actual segment count is 1 left shifted by the extracted value.

nSmplInSeg (Samples in a segment per one frequency band for the first channel set)

This is the number of samples in a segment per one frequency band in the first channel set. Representing a binary exponent, the actual number of samples in a segment is 1 left shifted by the extracted value. All subsequent channel sets determine their nSmplInSeg by first determining the sampling frequency scaling factor against the first channel set. For example, assume:

- the first channel set with the sampling rate Fs1 and the number of frequency bands equal to m_nNumFreqBands1 (see page 132 for the definition of m_nNumFreqBands);
- the second channel set with the sampling rate Fs2 and the number of frequency bands equal to m_nNumFreqBands2;
- then the second channel set will have the number of segment samples per frequency band equal to:

$$\text{nSmplInSeg} \times \frac{\text{Fs2} \times \text{m\_nNumFreq Bands1}}{\text{Fs1} \times \text{m\_nNumFreq Bands2}}$$

In order to control the required audio output buffer size, the maximum number of samples in a segment per each frequency band is limited as follows:

- Maximum nSmplInSeg is 256 for sampling frequencies Fs ≤ 48 kHz.
- Maximum nSmplInSeg is 512 for sampling frequencies Fs > 48 kHz.

NOTE: Notice that for sampling frequencies greater than 96 kHz, the data is split uniformly into 2 frequency bands (for 96 kHz < Fs ≤ 192 kHz) or into 4 frequency bands (for 192 kHz < Fs ≤ 384 kHz). Therefore the maximum sampling frequency in one frequency band is 96 kHz and consequently the maximum nSmplInSeg is the same for all sampling frequencies Fs > 48 kHz.

nBits4SSize (Number of bits used to read segment size)

nBits4SSize+1 is the bit size of the field that contains the size of all data fields in the NAVI table.

nBandDataCRCEn (Presence of CRC16 within each frequency band)

When set, this field indicates that checksums were embedded in the frequency band data. This field shall be decoded as follows:

Table 8-2: CRC Presence in Frequency Band

nBandDataCRCEn	CRC Presence
00	No CRC16 within band
01	CRC16 placed at the end of MSB0
10	CRC16 placed at end of MSB0 and at end of LSB0
11	CRC placed at end of MSB0 and at end of LSB0 and at the end of bands 1, 2 and 3 where they exist

bScalableLSBs (MSB/LSB split flag)

This indicates whether MSB/LSB split has been enabled. When bScalableLSBs is true, the MSB/LSB split has been performed in frequency band 0. The MSB/LSB split in extension frequency band nBand has been performed only when both bScalableLSBs and bMSBLSBSplitEnInExtBands[nBand] are true.

bOne2OneMapChannels2Speakers (Channels to speakers mapping type flag)

This flag is transmitted as a part of program descriptor within the extension substream header and its definition is repeated here for reference. This flag, if true, indicates that each encoded channel represents a signal feed to a corresponding loudspeaker on the decode site. If false, it indicates that channels carry the signals that describe the sound field but are not the actual loudspeaker feeds. The actual loudspeaker feeds are derived on the decode side using the stream embedded coefficients and possibly user-provided adjustment factors. One example of using this mode of operation is to carry Ambisonic first-order signals (B-format) W, X, Y and Z.

m_RepresentationType (Representation Type)

This field is transmitted as a part of program descriptor within the extension substream header and its definition is repeated here for reference.

The field is present if bOne2OneMapChannels2Speakers is false. It describes the type of representation according to the table below. This information may be useful in some post processing tasks. The decoder shall export this information to post-processing functions.

Table 8-3: Representation Types

m_RepresentationType	Description
0b000	Audio Asset for Mixing/Replacement
0b001	Reserved
0b010	L_t/R_t Encoded for matrix surround decoding implies nTotalNumCHs=2
0b011	L_h/R_h Audio processed for headphone playback implies nTotalNumCHs=2
0b100	Reserved
0b101 - 0b111	Reserved

nBits4ChMask (Channel Position Mask)

This is the number of bits used to extract the channel mask for each channel set. 'OR'-ing individual channel masks for all channel sets yields the overall channel mask for the frame.

nuFixedLSBWidth (MSB/LSB split)

This field only exists if bScalableLSBs is TRUE.

- if nuFixedLSBWidth = 0 then length of the LSB part is variable according to pnScalableLSBs[nBand][nCh];
- if nuFixedLSBWidth > 0 then length of the LSB part is fixed and equal to nuFixedLSBWidth.

In both cases the pnScalableLSBs[nBand][nCh] indicates the number of bits used to pack the binary codes representing the samples of LSB part.

Reserved (Reserved)

This is reserved for supplemental header information. The decoder shall assume that this field is present and of unspecified duration. Therefore, in order to continue unpacking the stream, the decoder shall skip over this field using the header start pointer and the header size **nHeaderSize**.

nCRC16Header (Header CRC16 Protection)

CRC16 is calculated for the header from positions VersionNum to ByteAlign inclusive.

8.3.2 Channel Set Sub-Header

Table 8-4: Channel Set Sub-Header

Syntax	Size (Bits)
`// Unpack the header size` `nByteOffset += m_nChSetHeaderSize = ExtractBits(10) + 1;`	10
`// Extract the number of channels` `m_nChSetLLChannel = ExtractBits(4) + 1;`	4
`// Extract the residual channel encoding` `m_nResidualChEncode = (DTS__int64) ExtractBits(m_nChSetLLChannel);`	nChSetLLChannel
`// Extract the input sample bit-width` `m_nBitResolution = ExtractBits(5) + 1;`	5
`// Extract the original input sample bit-width` `m_nBitWidth = ExtractBits(5) + 1;`	5
`// Extract the sampling frequency index` `sFreqIndex = ExtractBits(4);`	4
`// Find the actual sampling frequency` `m_nFs = m_pnFsTbl[sFreqIndex];`	
`// Extract nFs interpolation multiplier` `m_nFsInterpolate = ExtractBits(2);`	2
`// Extract the replacement channel set group` `m_nReplacementSet = ExtractBits(2);`	2

`// Extract the active replacement channel set flag`
`if (m_nReplacementSet > 0)`
`{`
`m_bActiveReplaceSet = (ExtractBits(1) == 1) ? true : false;`
`if (!m_bActiveReplaceSet)`
`m_bSkipDecode = true;`
`}`
	1

`// Downmix is allowed only when the encoded channel represents a signal //`
`feed to a corresponding loudspeaker (bOne2OneMapChannels2Speakers=true)`
`if (m_bOne2OneMapChannels2Speakers){`	

`// Extract the primary channel set flag` `m_bPrimaryChSet = (ExtractBits(1) == 1) ? true : false;`	1
`// Extract the downmix flag` `m_bDownmixCoeffCodeEmbedded = (ExtractBits(1) == 1) ? true : false;`	1

`// Extract the Embedded Downmix flag`
`if (m_bDownmixCoeffCodeEmbedded)`
`m_bDownmixEmbedded = (ExtractBits(1) == 1) ? true : false;`
	1

`// Extract the Downmix type`
`if (m_bDownmixCoeffCodeEmbedded && m_bPrimaryChSet)`
`m_nLLDownmixType = ExtractBits(3);`
	3

`// Extract the hierarchical channel set flag` `m_bHierChSet = (ExtractBits(1) == 1) ? true : false;`	1
`if (bDownmixCoeffCodeEmbedded)` `DownmixCoeffs = ExtractBits(nDownmixCoeffs * 9);`	nDownmixCoeffs×9
`bChMaskEnabled = (ExtractBits(1)==1)? true:false;`	1

`// Extract the ch mask`
`if (bChMaskEnabled)`
`{`
`m_nChMask = ExtractBits(nBits4ChMask);`
`}`
	nBits4ChMask

`if (!bChMaskEnabled) {`
`for (ch = 0; ch < nChSetLLChannel; ch++) {`
`RadiusDelta[ch] = ExtractBits(9);`
`Theta[ch] = ExtractBits(9);`
`Phi[ch] = ExtractBits(7);`
`}`
`}`
`}`
	25 per Ch

`else{ // Case when bOne2OneMapChannels2Speakers=false. No downmixing is`
`// allowed and each channel set is the primary channel set`
`bPrimaryChSet = true;`	

Syntax	Size (Bits)

m_bDownmixCoeffCodeEmbedded = false;
m_bHierChSet = true;
m_nNumChPrevHierChSet = 0;
m_nNumDwnMixCodeCoeffs = 0;
bMappingCoeffsPresent = ExtractBits(1);
	1

if (bMappingCoeffsPresent){
m_nBitsCh2SpkrCoef = ExtractBits(3);
// Map 0->6, 1->8 , … 7->20
	3

m_nBitsCh2SpkrCoef =6+2*m_nBitsCh2SpkrCoef m_nNumSpeakerConfigs = ExtractBits(2)+ 1;	2

nCofInd=0;
for (nSpkrConf =0; nSpkrConf <nNumSpeakerConfigs; nSpkrConf ++){
m_pnActiveChannelMask[nSpkrConf] = ExtractBits(nChSetLLChannel);
	nChSetLLChannel

m_pnNumSpeakers[nSpkrConf] = ExtractBits(6)+ 1;	6
bSpkrMaskEnabled = (ExtractBits(1)==1)? true:false;	1

// Extract the mask
if (bSpkrMaskEnabled)
m_nSpkrMask[nSpkrConf]=ExtractBits(nBits4ChMask);
	nBits4ChMask

for (nSpkr=0; nSpkr<m_pnNumSpeakers[nSpkrConf]; nSpkr++){
// Extract speaker locations in polar coordinates
if (!bSpkrMaskEnabled){
RadiusDelta[nSpkrConf][nCh] = ExtractBits(9);
Theta[nSpkrConf][nCh] = ExtractBits(9);
Phi[nSpkrConf][nCh] = ExtractBits(7);
}
// Extract channel to speaker mapping
// coefficients for each active channel
	25 per speaker

for (nCh=0; nCh< nChSetLLChannel; nCh++){
if (m_nActiveChannelMask[nSpkrConf]&(1<< nCh))
m_pnCh2SpkrMapCoeff[nCofInd++] =
ExtractBits(m_nBitsCh2SpkrCoef);
}
	m_nBitsCh2SpkrCoef per active channel

} // End speaker loop indexed by nSpkr
} // End speaker configuration loop indexed by nSpkrConf
} // if (bMappingCoeffsPresent)
} // End of bOne2OneMapChannels2Speakers=false case	

// Extract the num of frequency bands
if (m_nFs > 96000) {
bXtraFreqBands = ExtractBits(1);
if (bXtraFreqBands == 1)
m_nNumFreqBands = 4;
else
m_nNumFreqBands = 2;
}
else
m_nNumFreqBands = 1;
	1

// Unpack the pairwise channel decorrelation enable flag // for Band 0	1

nBand = 0;
bPWChDecorEnabled[nBand] = ExtractBits(1);
// Unpack the original channel order
for (ncBits4ChOrder = 0, n = 1; n < m_nChSetLLChannel; ncBits4ChOrder++, n *= 2){
NULL;
}	

if (bPWChDecorEnabled[nBand] == 1)
{
// Unpack the original channel order
for (nCh = 0; nCh < m_nChSetLLChannel; nCh++)
m_nOrigChanOrder[nBand][nCh] = ExtractBits(ncBits4ChOrder);
}
	Calculated

// The code for bChPFlag is embedded within nPWChPairsCoeffs below
// For nBand 0 unpack the channel pairwise flags and coefficients
if (bPWChDecorEnabled[nBand] == 1)
for (nCh = 0; nCh < (m_nChSetLLChannel >> 1); nCh++) {
bChPFlag[nBand] = ExtractBits(1);
if (bChPFlag[nBand] == 1){
nTmp = ExtractBits(7); // Unpack as unsigned
// Map to signed
	1

m_anPWChPairsCoeffs[nBand][nCh] =
(Tmp & 0x1) ? -(nTmp >> 1) - 1 : nTmp >> 1;
}
else
// Zero out coefficients when channel decorrelation is disabled
	7 per Ch pair (Conditional)

Syntax	Size (Bits)

`      m_anPWChPairsCoeffs[nBand][nCh] = 0;`
`  }`
`// Unpack the optimal adaptive predictor order;`
`// one per channel on the frame basis, 0 indicates no prediction`
`m_nCurrHighestLPCOrder[nBand] = 0;`
`for (nCh = 0; nCh < m_nChSetLLChannel; nCh++)`
`  {`
`    m_pnAdaptPredOrder[nBand][nCh] = ExtractBits(4);`
`    m_nCurrHighestLPCOrder[nBand] =`
`      (m_pnAdaptPredOrder[nBand] [nCh] >`
`       m_nCurrHighestLPCOrder[nBand]) ? m_pnAdaptPredOrder[nBand] [nCh] :`
`      m_nCurrHighestLPCOrder[nBand];`
`  }`
	4 per Ch

`// Unpack the optimal fixed predictor order per channel`
`// on the frame basis`
`// Only for channels where m_pnAdaptPredOrder[0][nCh]=0`
`// 0 indicates no prediction`
`for (nCh = 0; nCh < m_nChSetLLChannel; nCh++)`
`  {`
`    if (m_pnAdaptPredOrder[nCh] == 0)`
`      m_pnFixedPredOrder[0][nCh] = ExtractBits(2);`
`    else`
`      m_pnFixedPredOrder[0][nCh] = 0;`
`  }`
	2 per Ch (Conditional)

`// Unpack adaptive predictor quantized`
`// reflection coefficients`
`pnTemp = m_pnLPCReflCoeffsQInd[nBand];`
`for (nCh = 0; nCh < m_nChSetLLChannel; nCh++){`
`    for (n = 0; n < m_pnAdaptPredOrder[nBand][nCh]; n++){`
`        nTmp = ExtractBits(8);  // Unpack as unsigned`
`         // Map to signed`
`        *pnTemp++ = (nTmp & 0x1) ? -(nTmp >> 1) - 1 : nTmp >> 1;`
`    }`
`  }`
	8 bits per coefficient

`if (bScalableLSBs)` `  nLSBFsize[0] = ExtractBits(nBits4SSize);`	nBits4Ssize (Conditional)

`if (m_nBitWidth > 16)`
`  m_nBits4ABIT = 5;`
`else if (m_nBitWidth > 8)`
`  m_nBits4ABIT = 4;`
`else`
`  m_nBits4ABIT = 3; // for m_nBits4ABIT=5 case it has been taken care by`
`                    // LSB/MSB spliting`
`if (nNumChSetsInFrame > 1 && m_nBits4ABIT < 5)`
`  m_nBits4ABIT += 1; // to take care of ES saturation`
	4 per Ch (Conditional)

`if (bScalableLSBs){`
`  for (nCh = 0; nCh < m_nChSetLLChannel; nCh++)`
`    m_pnScalableLSBs[0][nCh] = ExtractBits(4);`
`}`
`else {`
`  for (nCh = 0; nCh < m_nChSetLLChannel; nCh++)`
`    m_pnScalableLSBs[0][nCh] = 0;`
`}`	

`if (bScalableLSBs){`
`  for (nCh = 0; nCh < m_nChSetLLChannel; nCh++)`
`    m_pnBitWidthAdjPerCh[0][nCh] = ExtractBits(4);`
`}`
`else {`
`  for (nCh = 0; nCh < m_nChSetLLChannel; nCh++)`
`    m_ pnBitWidthAdjPerCh [0][nCh] = 0;`
`}`
	4 per Ch (Conditional)

`// Extract extra frequency band parameters`
`for (nBand = 1; nBand < m_nNumFreqBands; nBand++){`
`  // Unpack the pairwise channel decorrelation enable`
`   // flag  for extension bands`
`  bPWChDecorEnabled[nBand] = ExtractBits(1);`
	1

`  // Unpack the original channel order`
`  if (bPWChDecorEnabled[nBand] == 1){`
`    for (nCh = 0; nCh < m_nChSetLLChannel; nCh++)`
`      m_nOrigChanOrder[nBand][nCh]=  ExtractBits(ncBits4ChOrder);`
`  }`
	Calculated

`  // The code for bChPFlag is embedded within` `  // nPWChPairsCoeffs below`	1 per Ch pair
`  // For an extension band unpack the channel pairwise` `  // flags and if true the corresponding coefficients`	7 per Ch pair (Conditional)

Syntax	Size (Bits)
`if (bPWChDecorEnabled[nBand] == 1){`	
`  for (nCh = 0; nCh < (m_nChSetLLChannel >> 1); nCh++){`	
`    bChPFlag[nBand] = ExtractBits(1);`	
`    if (bChPFlag[nBand] == 1){`	
`      nTmp = ExtractBits(7);  // Unpack as unsigned`	
`      m_anPWChPairsCoeffs[nBand][nCh] =`	
`       (nTmp & 0x1) ? -(nTmp >> 1) - 1 : nTmp >> 1; // Map to signed`	
`    }`	
`    else`	
`      // Zero out coefficients when channel decorrelation is disabled`	
`      m_anPWChPairsCoeffs[nBand][nCh] = 0;`	
`  }`	
`}`	
`// Unpack the optimal adaptive predictor order;`	
`// one per channel on the frame basis`	
`// 0 indicates no prediction`	
`m_nCurrHighestLPCOrder[nBand] = 0;`	
`for (nCh = 0; nCh < m_nChSetLLChannel; nCh++){`	4 per Ch
`  m_pnAdaptPredOrder[nBand][nCh] = ExtractBits(4);`	
`  m_nCurrHighestLPCOrder[nBand] =`	
`   (m_pnAdaptPredOrder[nBand][nCh] >  m_nCurrHighestLPCOrder[nBand]) ?`	
`    m_pnAdaptPredOrder[nBand][nCh]:m_nCurrHighestLPCOrder[nBand];`	
`}`	
`// Unpack the optimal fixed predictor order per`	
`// channel  on the frame basis`	
`// Only for channels where m_pnAdaptPredOrder[nCh]=0`	
`// 0 indicates no prediction`	
`for (nCh = 0; nCh < m_nChSetLLChannel; nCh++){`	2 per Ch (Conditional)
`  if (m_pnAdaptPredOrder[nBand][nCh] == 0)`	
`    m_pnFixedPredOrder[nBand][nCh] = ExtractBits(2);`	
`  else`	
`    m_pnFixedPredOrder[nBand][nCh] = 0;`	
`}`	
`// Unpack adaptive predictor quantized`	
`// reflection coefficients`	
`pnTemp = m_pnLPCReflCoeffsQInd[nBand];`	
`for (nCh = 0; nCh < m_nChSetLLChannel; nCh++){`	8* pnAdaptPredOrder[ch] per Ch
`  for (n=0; n<m_pnAdaptPredOrder[nBand][nCh]; n++){`	
`    nTmp = ExtractBits(8);  // Unpack as unsigned`	
`    *pnTemp++ = (nTmp & 0x1) ? -(nTmp >> 1) - 1 : nTmp >> 1;`	
`  }`	
`}`	
`if (m_bDownmixEmbedded)`	
`  m_bEmbDMixInExtBand[nBand] = ExtractBits(1);`	1 (Conditional)
`m_bMSBLSBSplitEnInExtBands[nBand] = ExtractBits(1);`	1 (Conditional)
`if (bMSBLSBSplitEnInExtBands[nBand])`	
`  nLSBFsize[nBand] = ExtractBits(nBits4SSize);`	nBits4Ssize (Conditional)
`else`	
`  nLSBFsize[nBand] = 0;`	
`if (bMSBLSBSplitEnInExtBands[nBand]){`	
`  for (nCh = 0; nCh < m_nChSetLLChannel; nCh++)`	
`    m_pnScalableLSBs[nBand][nCh] = ExtractBits(4);`	
`}`	4 per Ch (Conditional)
`else{`	
`  for (nCh = 0; nCh < m_nChSetLLChannel; nCh++)`	
`    m_pnScalableLSBs[nBand][nCh] = 0;`	
`}`	
`m_bFlagScalableResExtBand[nBand] = ExtractBits(1);`	1
`if (m_bFlagScalableResExtBand[nBand]){`	
`  for (nCh = 0; nCh < m_nChSetLLChannel; nCh++)`	
`    m_pnBitWidthAdjPerCh[nBand][nCh] = ExtractBits(4);`	
`}`	4 per Ch (Conditional)
`else{`	
`  for (nCh = 0; nCh < m_nChSetLLChannel; nCh++)`	
`    m_pnBitWidthAdjPerCh[nBand][nCh] = 0;`	
`}`	
`} // End Extra frequency band loop`	
`Reserved = ExtractBits(…);`	…
`ByteAlign = ExtractBits(0 … 7);`	0...7
`nCRC16SubHeader = ExtractBits(16);`	16

nChSetHeaderSize (Size of Channel Set Sub-header)

nChSetHeaderSize+1 is the size of the channel set sub-header in bytes. Use this to locate the sub-header of the next channel set.

nChSetLLChannel (Number of Channels in Set)

This indicates the number of channels in the channel set.

nResidualChEncode (Residual Type)

This is an array of bits (one bit per encoded channel) indicating the nature of the residual signal that is encoded in each of the encoded channels. For a particular encoded channel the value of corresponding bit in nResidualChEncode equal to '0' indicates that the residual in that channel is obtained by subtracting the lossy decoded audio from the original audio. For a particular encoded channel the value of corresponding bit in nResidualChEncode equal to '1' indicates that the residual in that channel is the original audio. Bits in the array nResidualChEncode are ordered according to the channel numbers where the highest channel number corresponds to the most significant bit.

nBitResolution (PCM Bit Resolution)

This expresses the original resolution of the PCM audio source as output by an A/D converter.

nBitWidth (Storage Unit Width)

This specifies the bit width of the storage media. For example, the PCM audio with nBitResolution=20 would be stored in wav files with the nBitWidth equal to 24. In the lossless decoder, in order to recreate the wav file that matches the original wav file, the decoded samples shall be shifted to the right by the amount equal to nBitWidth - nBitResolution.

sFreqIndex (original sampling frequency)

This is the sample rate of audio in a channel set. It corresponds to the resulting sampling frequency after interpolation by the nFsInterpolate factor, on the encode side. The 4-bit sFreqIndex field is interpreted as described in Table 8-5. Remember that certain extra fields are encountered in the data stream once the sample rate is greater than 96 kHz.

Table 8-5: sFreqIndex Sample Rate Decoding

sFreqIndex	Sample Frequency (kHz)
0	8 000
1	16 000
2	32 000
3	64 000
4	128 000
5	22 050
6	44 100
7	88 200
8	176 400
9	352 800
10	12 000
11	24 000
12	48 000
13	96 000
14	192 000
15	384 000

nFsInterpolate (sampling frequency modifier)

This specifies the sampling rate interpolation factor of the original signal. The sampling rate adjustment when mixing channel sets with different sampling rates, it specifies the interpolation factor applied to the original sampling frequency. This 2-bit code (see Table 8-6) encodes the Mvalue or Interpolation factor applied to the original sampling frequency.

Table 8-6: Sampling Rate Interpolation

nFsInterpolate	Mvalue
00	1
01	2
10	4
11	8

NReplacementSet

This is an indicator of which replacement set this set is a member of, (see Table 8-7). There is a maximum of 3 replacement sets.

Table 8-7: Replacement Set Association

nReplacementSet	Association
00	Not a replacement
01	Member of set 1
10	Member of set 2
11	Member of set 3

bActiveReplaceSet (Default replacement set)

This is present only if nReplacementSet != 0. In particular, bActiveReplaceSet =true indicates that the current channel set is the active channel set.

bPrimaryChSet

This is present only if bOne2OneMapChannels2Speakers is true. The bPrimaryChSet=true indicates that the set is a primary channel set. The primary channel sets represent the subset of all encoded channels, i.e. it contains data for 5.1 audio or a stereo downmix.

bDownmixCoeffCodeEmbedded (Downmix coefficients present in stream)

This is present only if bOne2OneMapChannels2Speakers is true. It indicates whether a matrix of downmix coefficients has been defined and is embedded in the stream.

bDownmixEmbedded (Downmix already performed by encoder)

This is present only if bOne2OneMapChannels2Speakers is true. If bDownmixCoeffEmbedded is set, then extract bDownmixEmbedded. When bDownmixEmbedded=true this indicates to the decoder that on the encode side, audio in the channels of the current channel set has been downmixed to channels in the lower channels sets indexed. After decoding the current channel set, the above mentioned encoder downmix operation needs to be undone in the decoder. If bDownmixEmbedded=false, the encoder did not perform the downmixing operation on the current channel.

nLLDownmixType

This is present only if bOne2OneMapChannels2Speakers is true. It indicates the downmix processing type for the primary ChSet group. The downmix action is defined in Table 8-8.

Table 8-8: Downmix Type

nLLDownmixType	Downmix primary ChSet group to
000	1/0
001	Lo/Ro
010	Lt/Rt
011	3/0
100	2/1
101	2/2
110	3/1
111	Unused

BHierChSet

This is present only if bOne2OneMapChannels2Speakers is true. It indicates whether the channel set is part of a hierarchy.

DownmixCoeffs

This is present only if bOne2OneMapChannels2Speakers is true. If bDownmixCoefCodeEmbedded SetDownmix = true, then extract nDownmixCoeffs. To extract the coefficients, the decoder needs to determine the number of downmix coefficients (NDownmixCoeffs) by calculating the size of the NxM table of coefficients. N is defined as nChSetLLChannel+1 rows, where the extra row represents the down scaling coefficients that prevent overflow. Conversely, the number of columns M (the number of channels that the current channel set is mixed into) is determined when the decoder decodes the channel set hierarchy down to the primary channel set.

Consider the case where there are 2 extended channel sets and 1 primary channel set (a total of 3 channel sets). When the decoder is currently unpacking the downmix coefficients defined in the second extended channel set (4 channels), N is 4+1=5. To determine M, the decoder shall traverse the channel set hierarchy and count the number of channels within each channel set. Hence, there are 6 channels in the primary set and 2 channels in the first extended channel set. Therefore M=8 and thus 5*8 coefficients.

When downmix coefficients are defined for the primary channel group, use the nLLDownmixType to determine M (number of resultant fold down channels) and the total number of channels in the primary channel group to determine N.

Coding of the downmix coefficients is described in clause C.9.

bChMaskEnabled (Channel Mask Enabled)

This is present only if bOne2OneMapChannels2Speakers is true. When set, it indicates that the channel set is using a predefined channel mask. Otherwise a user specified polar coordinate speaker configuration is applied.

nChMask (Channel Mask for Set)

This is present only if bOne2OneMapChannels2Speakers is true. When bChMaskEnabled is set, this field indicates which pre-defined channel positions apply. (See clause C.8 for details.)

ChSetSpeakerConfiguration (Angular Speaker Position Table)

This is present only if bOne2OneMapChannels2Speakers is true. If bChMaskEnabled==false, extract the speaker/channel configuration for the channel set. Each speaker (or channel) location is expressed in spherical coordinates (delta, theta, phi), where the origin is the location of the listener. The centre of each speaker is ideally located on the surface of the sphere surrounding the listener. Differences in speaker radii are stored as deltas from this imaginary spherical surface. The radius has a range [-510, 510] with resolution of 2 cm (9 bits). The angle theta is expressed as [-180, 180] with a resolution of 1 degree (9 bits). The angle phi is [-90, 90] with a resolution of 2 degrees (7 bits). The speakers each correspond to channels defined in the previous channel reference. After all objects have been extracted, the master speaker configuration can be created. The first speaker in the channel set defines the spherical surface and all other speakers have radii expressed as deltas relative to this first channel.

bMappingCoeffsPresent (Mapping Coefficient Present Flag)

The bMappingCoeffsPresent is present only if bOne2OneMapChannels2Speakers is false. When bMappingCoeffsPresent is true, it indicates that channel-to-speaker mapping coefficients are present in the stream.

When the bOne2OneMapChannels2Speakers is false, the encoded channels carry the signals that describe the sound field but are not necessarily the actual loudspeaker feeds. The actual loudspeaker feeds are derived on the decode side using the channel-to-speaker mapping coefficients and possibly user-provided adjustment factors. For certain types of representation, the channel-to-speaker mapping coefficients are not needed (i.e. L_h/R_h) or channel-to-speaker mapping coefficients are provided by some other metadata fields (like mixing data for audio assets). In those cases, the parameter bMappingCoeffsPresent may be set to false. Consequently no channel-to-speaker mapping coefficients and associated bit fields will be present in the stream.

nBitsCh2SpkrCoef

This is present only if bOne2OneMapChannels2Speakers is false and bMappingCoeffsPresent is true. This parameter represents a code for a number of bits used to pack each channel-to-speaker mapping coefficient. The actual number of bits is obtained from mapping:

$$m\_nBitsCh2SpkrCoef = 6 + 2 * m\_nBitsCh2SpkrCoef$$

nNumSpeakerConfigs (Number of Loudspeaker Configurations)

This is present only if bOne2OneMapChannels2Speakers is false and bMappingCoeffsPresent is true. Generally, the signals that describe the sound field- and are transmitted in a stream as channels-may be used to create speaker feeds for an arbitrary 3D speaker configuration. In some cases there exist exact mathematical relationships between the coefficients for different loudspeaker configurations. However, the content creators may wish to perform small adjustments to these relationships. In that case, it is beneficial to transmit separate sets of channel-to-speaker mapping coefficients for different speaker configurations (e.g. one set for 5.1 and one set for 7.1). The parameter nNumSpeakerConfigs indicates this number of different speaker configurations and consequently, the number of sets of channel-to-speaker mapping coefficients.

pnActiveChannelMask (Active channel mask for current loudspeaker configuration)

This is present only if bOne2OneMapChannels2Speakers is false and bMappingCoeffsPresent is true. This bit mask indicates the activity of each channel in the particular speaker configuration mapping. The least significant bit corresponds to channel 0 and the most significant bit corresponds to channel nChSetLLChannel -1. For some speaker configurations, not all of the encoded channel signals are used in channel-to-speaker mappings. For example, in Ambisonic B-format there are four signals (W, X, Y and Z) that would be coded as four channels. The signal Z, in particular, carries information about the sound field component along the z-axis in Cartesian coordinates. Therefore, mapping to a standard 5.1 loudspeaker layout would not involve the Z channel. Consequently the mask would be 0111.

pnNumSpeakers

This is present only if bOne2OneMapChannels2Speakers is false and bMappingCoeffsPresent is true. This parameter represents a number of speakers in a current loudspeaker configuration.

bSpkrMaskEnabled

This is present only if bOne2OneMapChannels2Speakers is false and bMappingCoeffsPresent is true.

Each of the nNumSpeakerConfigs speaker configurations, for which the channel-to-speaker mapping coefficients are defined, may be described using either:

- the predefined loudspeaker mask (bSpkrMaskEnabled is true); or
- the speaker positions in polar coordinates (bSpkrMaskEnabled is false).

nSpkrMask (Speaker mask for current loudspeaker configuration)

This is present only if bOne2OneMapChannels2Speakers is false, bMappingCoeffsPresent is true and the bSpkrMaskEnabled is true. It indicates which pre-defined loudspeaker positions define the current speaker configuration. See clause C.8 for details. One of the available configurations shall be a standard 5.1 layout (C, L, R, Ls, Rs, LFE1).

ChSetSpeakerConfiguration (Angular speaker position table for current loudspeaker configuration)

This is present only if bOne2OneMapChannels2Speakers is false and bMappingCoeffsPresent is true. If bSpkrMaskEnabled is false, extract the speaker position for the current loudspeaker configuration. Coding of loudspeaker location parameters is described in clause C.4.

pnCh2SpkrMapCoeff (Channel to loudspeaker mapping coefficients)

This is present only if bOne2OneMapChannels2Speakers is false and bMappingCoeffsPresent is true.

The pnCh2SpkrMapCoeff represents the set of channel-to-speaker mapping coefficients, one for each speaker of each loudspeaker configuration and for each active channel. Coefficients are represented as signed numbers in Q3.(m_nBitsCh2SpkrCoef-4) fixed point format, i.e. the actual coefficients in range from -8 to 8 are related to the transmitted coefficient pnCh2SpkrMapCoeff [n] as $\frac{\text{pnCh2SpkrM apCoeff[n]}}{2^{(\text{m\_nBitsCh 2SpkrCoef - 4})}}$. The mapping equation for the loudspeaker S_k and active channels Ch_j , $j=0,1, \ldots, M$ is as follows:

$$S_k = \sum_{j=0}^{j=M} Coeff_{kj} \cdot Ch_j$$

where the $Coeff_{kj}$ denotes the coefficient corresponding to the mapping from j-th active channel to the k-th loudspeaker in the current loudspeaker configuration.

bXtraFreqBands (Indicates extra frequency bands when the sample rate is greater than 96 kHz)

If the sample frequency (m_nFs) is greater than 96 kHz, then this boolean flag exists and it indicates the number of extra frequency bands present in the stream.

A value of '1' indicates that full bandwidth is preserved. In particular, the number of frequency bands that are present in the stream (m_nNumFreqBands) is:

$$\text{m\_nNumFreqBands} = \begin{cases} 2 & \text{for } 96 \text{ kHz} < \text{m\_nFs} \le 192 \text{ kHz} \\ 4 & \text{for } 192 \text{ kHz} < \text{m\_nFs} \le 384 \text{ kHz} \end{cases}$$

A value of '0' indicates that only one-half of the original bandwidth is preserved. In this case:

$$\text{m\_nNumFreqBands} = \begin{cases} 1 & \text{for } 96 \text{ kHz} < \text{m\_nFs} \le 192 \text{ kHz} \\ 2 & \text{for } 192 \text{ kHz} < \text{m\_nFs} \le 384 \text{ kHz} \end{cases}$$

For sample frequencies below or equal to 96 kHz, this field is not present and the number of frequency bands is one by default.

bPWChDecorEnabled[0] (Pairwise Channel Decorrelation for frequency band 0)

When this is set, one or more channel pairs have been processed with pairwise channel decorrelation. This type of processing is performed separately for each frequency extension band.

nOrigChanOrder[0] (Original channel order for frequency band 0)

If bPWChDecorEnabled[0] is set, unpack the original channel order for each channel in the channel set. If bPWChDecorEnabled[0] is set (at the encoder), the input channels have been grouped into pairs for the channel decorrelation process. In this case the original channel order is included in the bit stream. After lossless residual decoding, the original channel order shall be restored before either combining with the lossy output or outputting the residual itself when no core is present. The width of this field is calculated based on examining the first bit set size of nChSetLLChannel.

bChPFlag[0] (Channel pairwise flags in frequency band 0)

If bPWChDecorEnabled[0] is set, unpack the channel pairwise flags for each channel in the set. If bChPFlag is set, then a coefficient is included in the stream.

nPWChPairsCoeffs[0] (Pairwise Channel Coefficients for frequency band 0)

If bChPFlag for channel is set, unpack the pairwise decorrelation coefficients for that channel. This coefficient should be converted to a signed number and used to scale the source channel before adding the scaled version to the destination channel.

pnAdaptPredOrder[0] (Adaptive predictor order in frequency band 0 for each channel)

This is an adaptive predictor order per channel. A zero indicates no prediction.

pnFixedPredOrder[0] (Fixed predictor order for frequency band 0)

This is a fixed predictor order per channel in frequency band 0. A zero indicates no prediction.

pnLPCReflCoeffsQInd[0] (Adaptive predictor quantized reflection coefficients in frequency band 0)

This is present only if the adaptive prediction order is not zero.

nLSBFsize[0] (Size of the LSB section in any segment of frequency band 0)

When bScalableLSBs is set, the size of the LSB data, which is the same in all segments of frequency band 0, will be present in the stream. Since the size of the entire channel set in a particular segment of frequency band 0 is already defined in the NAVI table, the difference will indicate the size of the MSB section for that segment.

pnScalableLSBs[0] (Number of bits used to represent the samples in LSB part of frequency band 0; one per channel)

If bScalableLSBs is set, extract the pnScalableLSBs[0][nCh] for each channel nCh in a channel set. This is the number of bits used for representing the samples in LSB part of frequency band 0 in channel nCh.

pnBitWidthAdjPerCh[0] (Number of bits discarded by authoring in frequency band 0)

This is the number of bits that an authoring tool discarded from the frequency band 0 for bit rate management. If bScalableLSBs is set, extract pnBitWidthAdjPerCh for each channel in the channel set. Normally, in the encoded master, this field has value 0. After bit rate management, this value is used to pad out the decoded audio to ensure the number formats of all decoded audio are alike before downmix reversal.

bPWChDecorEnabled[nBand] (Pairwise Channel Decorrelation for frequency extension band nBand)

This is present only when m_nNumFreqBands>1. When set for a particular frequency extension band, one or more channel pairs have been processed with pairwise channel decorrelation. This type of processing is performed separately for each frequency extension band.

nOrigChanOrder[nBand] (Original channel order for extension frequency band)

If bPWChDecorEnabled[nBand] is set, unpack the original channel order for each channel in the channel set. If bPWChDecorEnabled[nBand] is set (at the encoder), the input channels have been grouped into pairs for the channel decorrelation process. In this case the original channel order is included in the bit stream. After lossless residual decoding, the original channel order shall be restored before either combining with the lossy output or outputting the residual itself when no core is present. The width of this field is calculated based on examining the first bit set size of nChSetLLChannel.

bChPFlag[nBand] (Channel pairwise flags in frequency extension band nBand)

When the m_nNumFreqBands>1 and when the bPWChDecorEnabled[nBand] is set, unpack the channel pairwise flags for each channel in the set.

nPWChPairsCoeffs[nBand] (Pairwise Channel Coefficients for frequency extension band nBand)

When the m_nNumFreqBands>1 and when bChPFlag[nBand[nCh] for channel nCh is set, unpack the pairwise decorrelation coefficients for that channel. This coefficient should be converted to a signed number and used to scale the source channel before adding the scaled version to the destination channel.

pnAdaptPredOrder[nBand] (Adaptive predictor order for extra frequency band nBand)

This is present only when m_nNumFreqBands>1. It indicates the adaptive predictor order per channel. A zero indicates no prediction.

pnFixedPredOrder[nBand] (Fixed predictor order for extra frequency band nBand)

This is present only when m_nNumFreqBands>1 and the pnAdaptPredOrder[nBand][nCh]=0. It indicates the fixed predictor order per channel nCh in extra frequency band nBand. A zero indicates no prediction.

pnLPCReflCoeffsQInd[nBand] (Adaptive predictor quantized reflection coefficients in extra frequency band nBand)

This is present only if the adaptive prediction order is not zero.

bEmbDMixInExtBand[nBand] (MSB/LSB split flag in extension frequency band)

This is present only when the m_nNumFreqBands>1 and the bDownmixEmbedded is true. When the bEmbDMixInExtBand[nBand] is true, it indicates that the embedded downmix has been performed in frequency band nBand. If bEmbDMixInExtBand is false for all extension frequency bands, then the downmix is embedded only in frequency band 0 and the samples in extension frequency bands are scaled accordingly.

bMSBLSBSplitEnInExtBands[nBand] (MSB/LSB split flag in extension frequency band)

This is present only when the m_nNumFreqBands>1 and it indicates whether the MSB/LSB split has been performed in frequency band nBand.

nLSBFsize[nBand] (Size of the LSB section in extension frequency band nBand)

This is present only when the bMSBLSBSplitEnInExtBands[nBand] is true and it indicates the size of the LSB data, which is the same in all segments of one frequency band. Since the size of the entire channel set in a particular segment of a frequency band (indexed by nBand) is already defined in the NAVI table, the difference will indicate the size of the MSB section in a particular segment of the nBand-th frequency band.

pnScalableLSBs[nBand] (Number of bits used to represent the samples in LSB part of frequency band nBand; one per channel)

If bMSBLSBSplitEnInExtBands[nBand] is true, extract the pnScalableLSBs[nBand][nCh] for each channel nCh in a channel set. This is the number of bits used for representing the samples in the LSB part of frequency band nBand in channel nCh.

bFlagScalableResExtBand[nBand] (Scalable Resolution in Extension Band Enable Flag)

When the m_nNumFreqBands>1 and when the bFlagScalableResExtBand is set, the reduction of bit width of extension band data may be performed during the authoring process in order to reduce the bit rate requirements.

pnBitWidthAdjPerCh[nBand] (Number of bits discarded by authoring in extra frequency band nBand)

This field is present only in the case when m_nNumFreqBands>1 and bFlagScalableResExtBand[nBand] is true. This is the count of the number of bits that an authoring tool discarded for bit-rate management from the extra frequency band nBand. Notice that unlike the case of frequency band 0, where the bit width reduction is only allowed in the LSB part of the data, the bit width reduction of extension frequency band data may occur in LSB as well as the MSB parts. Consequently, in extension frequency bands the bit-rate management may be performed even when the MSB/LSB split is not enabled (bFlagScalableResExtBand[nBand] is false).

For each extra frequency band (nBand), read pnBitWidthAdjPerCh[nBand] for each channel in the channel set. Normally, in the encoded master this field has value 0. After bit-rate management, this value is used to pad out the decoded audio to ensure the number formats of all decoded frequency bands are alike before final filter bank interpolation.

Reserved (Reserved)

This is reserved for supplemental channel set header information. The decoder shall assume that this field is present and of unspecified duration. Therefore in order to continue unpacking, the stream decoder shall skip over this field using the channel set header start pointer and the channel set header size nChSetHeaderSize.

nCRC16SubHeader (CRC16 of channel set sub-header)

This is the CRC16 of the entire channel set header from positions nChSetHeaderFSize to ByteAlign inclusive.

8.3.3 Navigation Index Table

The navigation index table, or NAVI, is comprised of the sizes of individual frequency bands. It is described in more detail in clause 8.4.2.

8.3.4 Frequency Bands

Table 8-9 details the composition of the frequency bands.

Table 8-9: Frequency Bands

Syntax	Size (Bits)
FreqBandData = ExtractBits(var);	Var
ByteAlign = ExtractBits(0 … 7);	0...7

FreqBandData (Frequency Band Data)

This contains the frequency band data.

8.4 Lossless Stream Synchronization & Navigation

8.4.1 Overview of XLL Navigation

Intrinsic features of the DTSXLL stream format are its synchronization, navigation, error handling and recovery abilities. The following issues are addressed:

- Navigation Index
- Stream Navigation
- Error Detection
- Error Resilience

8.4.2 Navigation Index

The navigation index (NAVI) depicted in Figure 8-4 contains the size of each of the individual components contained in the lossless stream. The number of entries in the NAVI table is determined by the number of frequency bands, the number of segments and the number of channel sets. Also note that NOT all channel sets will have components for all frequency bands. As a result, the number of frequency bands designated for each channel set sub-header should be used in the calculation of the overall number of entries. Additionally the *nBits4Ssize* field globally determines the bit width of each individual entry in the index.

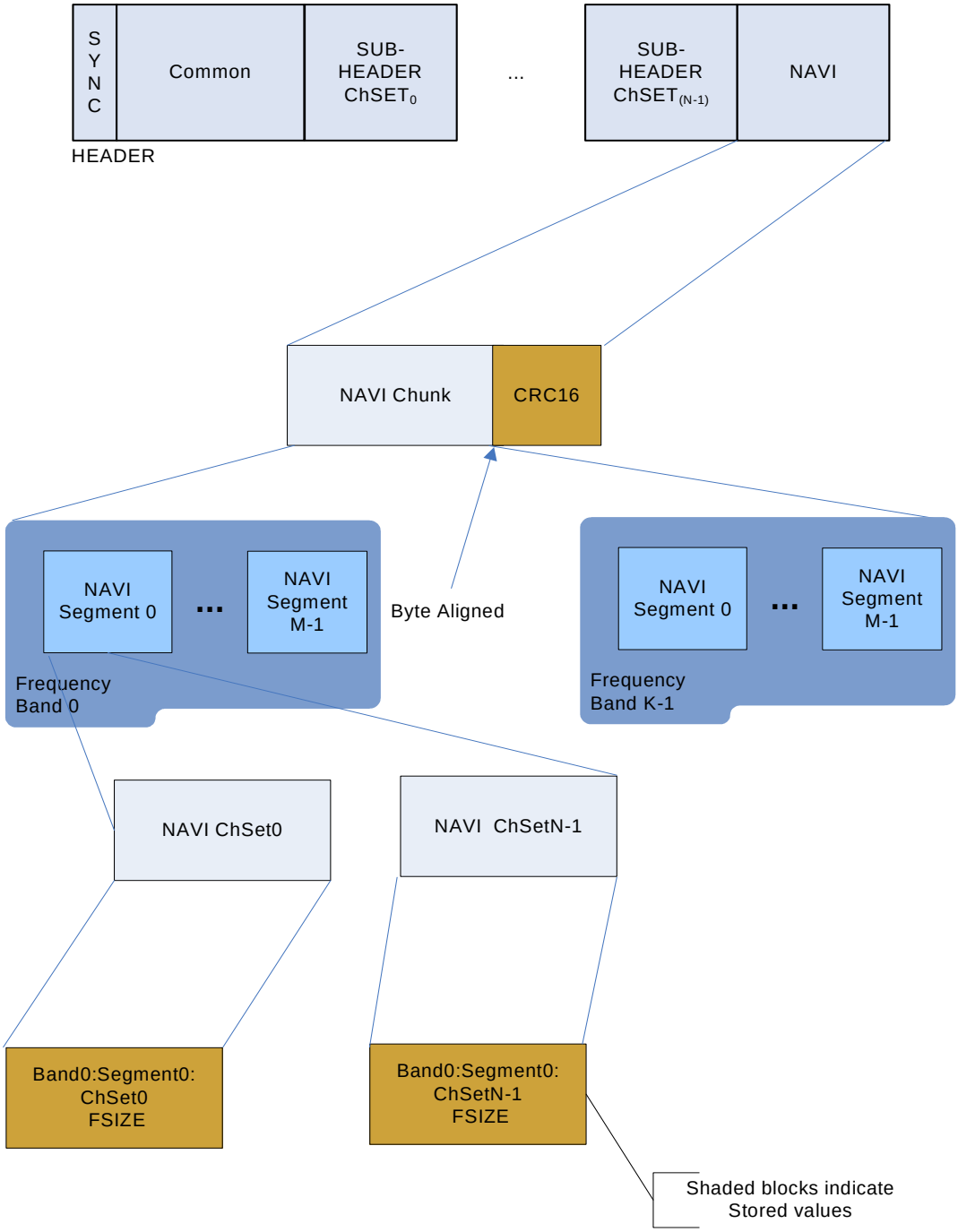


Figure 8-4: Navigation Index

Since the NAVI data is essential to reliable decoding of the frame and error recovery, the data is protected by a CRC16 checksum. It is easy to see that random access to any point in the stream can be achieved by the summation of prior entries in the index. For example, pseudo-code to illustrate the index packing order is given below.

```
M = number_of_segments;
N = number_of_channelsets;
P = number_of_frequency_bands;

for (Band=0; Band<P; Band++){
    BandSize[Band]=0;
    for(Seg=0;Seg<M;Seg++){
        SegmentSize[Band][Seg]=ExtractBits(nBits4Ssize);
        for(ChSET=0; ChSET<N; ChSET++){
            if ( GetNumFreqBand4ChSet[ChSET] > Band ){
                BandChSetSize[Band][Seg][ChSET]=ExtractBits(nBits4Ssize)+1;
                SegmentSize[Band][Seg] += BandChSetSize[Band][Seg][ChSET];
            }
```

```
            }
            BandSize[Band] += SegmentSize[Band][Seg];
        }
    }
    // Re-align bit pointer to byte position
    bitptr = next_byte_position_in_stream();
    Checksum = ExtractBits(16);
```

The NAVI CRC16 is computed for all data expressed in the listing above, inclusive of any byte alignment fields. When byte alignment is applied, the value inserted in the stream shall be zero.

8.4.3 Stream Navigation

Navigation to the checksums in both the header and channel set sub header is achieved by reading the size and subtracting the width of the checksum field. After offsetting the pointer from the present position, the checksum field is then immediately found.

Navigation to the start position of the first segment in the frame is obtained by summing the size of the header, the size of each channel set sub-header and the computed size of the NAVI table.

At the segment level, all channel sets are packed in numerically-increasing sequential order. For instance, the first channel set in the segment is the primary channel set (5.1) and the second set in the segment contains the extended channel (ES) set.

Navigation from one frequency band to the next sequential frequency band is achieved by computing the sum of all bands (or all segments) within the current frequency band and advancing by this offset to the next frequency band.

Navigation from one segment of a frequency band to the next sequential segment in the same frequency band is achieved by computing the sum of all bands for all channel sets in the present segment set and advancing by this offset to the next segment.

Navigation from one packed channel set to the next sequential channel set in the same frequency band and segment is achieved by using the band size of the current channel set and advancing by this offset to the next channel set.

Navigation within a band to the start of an LSB band data is achieved by subtracting the LSB band size located in the channel set sub-header from the frequency band size indicated in the navigation index.

8.4.4 Error Detection

Checksums are the principle mode for verifying that data is intact. Error detection is assisted by the presence of CRC fields at key points within the lossless data stream. The CRC fields validate the data just read, but they also play a significant role in verifying that the stream synchronization is correct. The checksum of the headers is handled as described below. Note that when a checksum is included in the validation check, the result is zero.

The common headers, channel set sub-header and NAVI table are protected by performing a CRC16 checksum generation across all the header data, inclusive of byte alignment. When the checksum is included as part of the evaluation range, the output from the CRC16 detection algorithm will be zero if no fault condition is present. Any non-zero value indicates a fault.

The frequency bands can be checked for errors in two ways:

- First, by ensuring that-after unpacking-the bit position travelled expressed as a whole number of bytes correlates with the band size extracted from NAVI; and/or
- Second, by computing a CRC16 for the entire frequency band data and comparing with an optional stored CRC16 checksum. This checksum when flagged by *nBandDataCRCEn* could be positioned in the frequency band data at one of three locations:

    1) the end of MSB0;

    2) the end of MSB0 and at end of LSB0; or

    3) the end of MSB0 and at end of LSB0 and at the end of bands 1, 2 and 3, where they exist.

Each frequency band terminates on a byte boundary and is zero padded until this condition is met. The optional checksum (CRC16) may then be placed at this position. The value stored in the NAVI table will be representative of the size for all frequency band data and byte alignments and will indicate whether the checksum is present or not.

8.4.5 Error Resilience

When an error condition has occurred, the decoder behaviour shall be according to Table 8-10 for the fault conditions listed.

Table 8-10: Error Handling

Fault	Solution
Header is corrupt	Advance to the next frame by returning to sync detection
Any channel set header is corrupt	Advance to the next frame by returning to sync detection
NAVI is corrupt	Advance to the next frame by returning to sync detection
Invalid entry in any header field	Advance to the next frame by returning to sync detection
1st frequency band data is corrupt	Advance to next segment if possible. Otherwise, advance to the next frame
2nd frequency band data is corrupt	Render the 1st frequency band and zero all subsequent frequency band data
3rd frequency band data is corrupt	Render the 1st and 2nd frequency bands and zero all subsequent frequency band data
4th frequency band data is corrupt	Render the 1st, 2nd and 3rd frequency bands and zero the 4th frequency band data
Any other fault condition	Advance to the next frame by returning to sync detection

8.5 Lossless Stream Decoding

8.5.1 Overview of Lossless Decoding

The data stream input to the decoder consists of both lossy (core) and lossless content. If the stream contains both lossy and lossless components, a core decoder will decode only the lossy part, whereas a lossless decoder will reconstruct the residual signal and combine it with the lossy part before output. If the stream does not contain a backward-compatible lossy stream, then only the lossless decoder will be active.

There can be additional frequency band data that contain information for increasing the sample rate. The total number of frequency bands is deduced from the field bXtraFreqBands, which is only present in the channel set sub-header when sFreqIndex is greater than 96 kHz. When this field is set, it will indicate that the total number of frequency bands (m_nNumFreqBands) is:

$$\text{m\_nNumFreqBands} = \begin{cases} 2 & \text{for } 96\ \text{kHz} < \text{m\_nFs} \le 192\ \text{kHz} \\ 4 & \text{for } 192\ \text{kHz} < \text{m\_nFs} \le 384\ \text{kHz} \end{cases}$$

Otherwise, if clear, it indicates that the total number of frequency bands is:

$$\text{m\_nNumFreqBands} = \begin{cases} 1 & \text{for } 96\ \text{kHz} < \text{m\_nFs} \le 192\ \text{kHz} \\ 2 & \text{for } 192\ \text{kHz} < \text{m\_nFs} \le 384\ \text{kHz} \end{cases}$$

The following diagram illustrates how to unpack frequency band 0 data.

A decoder capable of decoding the lossless extension (Figure 8-5) will perform the following tasks:

- Unpack frequency band 0 data - stream parsing.
- Inverse fixed prediction - linear signal reconstruction.
- Inverse adaptive prediction - adaptive signal reconstruction.
- Inverse pairwise channel de-correlation - scaling and reordering of channels.

- Reconstruct PCM samples - summation of core and lossless samples.

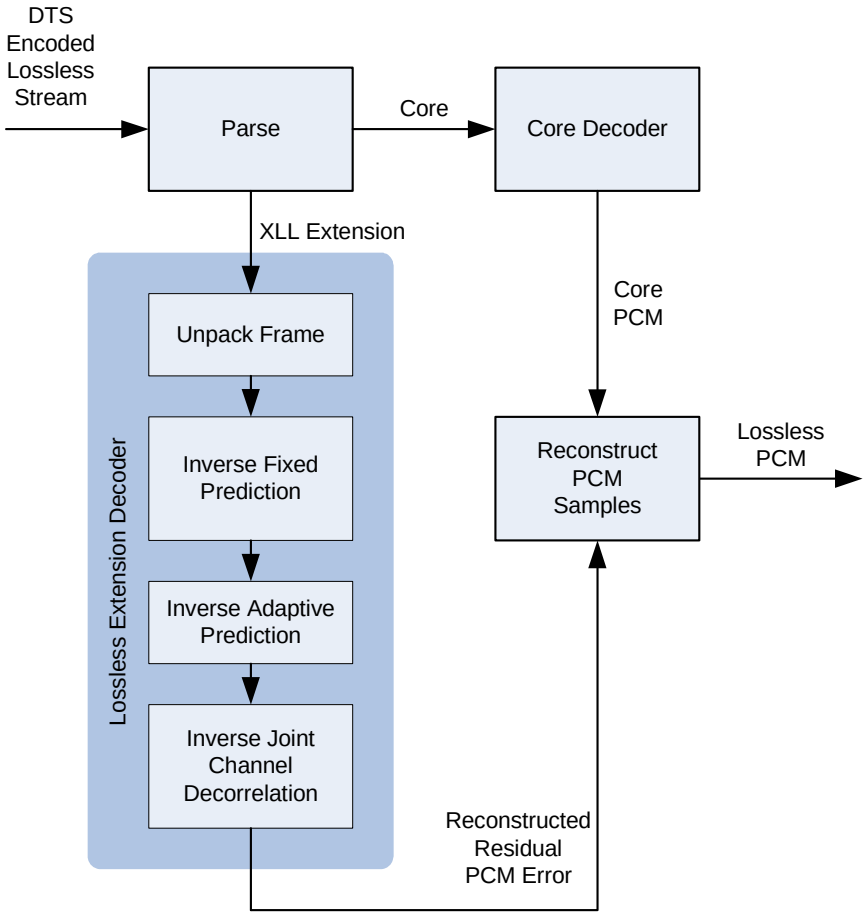


Figure 8-5: DTS Lossless Decoder

8.5.2 Band Data

8.5.2.1 General Information About Band Data

By default, the first band in a channel set contains the encoded data for sampling frequencies less than or equal to 96 kHz.

There are two types of data packing, as indicated by ncSegtype:

- Type 0, for which the coding parameters (bit allocation, etc.) are unique for each channel in the frequency band. The block of Rice code flags for each channel in the channel set comes first. The block of corresponding coding parameters for each channel in the channel set comes next. This is followed by the block of entropy codes for all residual samples in the segment for all channels of the channel set.

- Type 1, for which all channels in the frequency band use a common set of coding parameters. The Type 1 segment is unpacked in a similar way to Type 0, except that for all channels in the channel set, a common Rice code flag and the common ABIT value(s) appear in the stream before the block of entropy codes.

Regardless of the data packing type, the data in segment 0 is further sub-divided into two parts:

- Part 1 - Start of PCM.

- Part 2 - ADPCM.

To ensure that frames can be edited independently, ADPCM history is not carried between the frames. Instead, at the beginning of each frame (in the Part 1 section of segment 0), adaptive prediction is disabled and re-enabled only after a sufficient number of PCM samples are available for initialization of the predictor's delay line. The prediction residuals obtained after enabling adaptive prediction constitute the Part 2 section. Only segment 0 is permitted to have both Part 1 and Part 2 sections. All other segments shall be entirely comprised of Part 2 data. The number of samples in Part 1 is dependent on the order of the adaptive predictor that is used in each particular channel and the segment type (ncSegtype). In particular, the number of samples in Part 1 of segment 0 is determined as follows:

- ncSegtype = 0; pnAdaptPredOrder[nCh] for nCh=0,… nChSetLLChannel;
- ncSegtype = 1; m_nCurrHighestLPCOrder for all channels in the current channel set;

where nCurrHighestLPCOrder denotes the highest ADPCM order among all the channels of the current channel set.

8.5.2.2 Unpacking Frequency Band Data

The data in all bands is packed according to the same stream syntax.

The following flowcharts and code detail the syntax of the band data whose composition dynamically varies as the data is read.

The band data can consist of four formats:

- Linear-encoded MSBs.
- RICE-encoded MSBs.
- "Hybrid" RICE-encoded MSBs.
- Linear-encoded LSBs.

The following pseudo-code illustrates this procedure:

```
// Start unpacking MSB portion of segment
if ( nSeg != 0 )
    bUseSegStateCodeParm = (ExtractBits(1) == 1 ) ? true : false;
// Unpack segment type:
// - 0 --> implies use of distinct coding parameter for each channel
// - 1 --> implies use of common coding parameter for all channel
if ( ! bUseSegStateCodeParm )
    m_ncSegType = ExtractBits(1);
// Determine num of coding parameter sets encoded in segment
// For segment type == 0, distinct coding parameter for each channel
// For segment type == 1, same coding parameter for all channel
nNumParmSets = (m_ncSegType == 0) ? m_nChSetLLChannel : 1;

if ( ! bUseSegStateCodeParm )
{
// Allocate resources to store coding parameters
    ReallocCodingParm(nNumParmSets);
    // Unpack Rice coding flag false->linear code; true-> Rice code
    for ( nParmIndex = 0; nParmIndex < nNumParmSets; nParmIndex++ ){
        m_pabRiceCodeFlag[nParmIndex] = (ExtractBits(1) == 1) ? true : false;
        if (m_ncSegType==0 && m_pabRiceCodeFlag[nParmIndex]==true){
            // Unpack Hybrid Rice coding flag 0->Rice code; 1-> Hybrid Rice code
            if (ExtractBits(1) == 1)
            // Unpack binary code length for isolated max samples in Hybrid Rice coding
                m_pancAuxABIT[nParmIndex] = ExtractBits(m_nBits4ABIT)+1;
            else
                m_pancAuxABIT[nParmIndex] = 0; // 0 indicates no Hybrid Rice coding
        }
    }
    // Unpack coding parameter
    for ( nParmIndex = 0; nParmIndex < nNumParmSets; nParmIndex++ )
    {
        if (nSeg==0)
        {
            // Unpack coding parameter for part 1 of segment 0
            m_pancABIT0[nParmIndex] = ExtractBits(m_nBits4ABIT);
            if (m_pabRiceCodeFlag[nParmIndex]==false) // Adjustement for the linear code
            m_pancABIT0[nParmIndex] = (m_pancABIT0[nParmIndex]==0) ? 0 : m_pancABIT0[nParmIndex]+1;
            if ( m_ncSegType == 0 )
```

```
                m_panSamplPart0[nParmIndex] = m_pnAdaptPredOrder[nParmIndex];
            else
                m_panSamplPart0[nParmIndex] = m_nCurrHighestLPCOrder;
        }
        else
            m_panSamplPart0[nParmIndex] = 0;

        // Unpack coding parameter for part 2 of segment
        m_pancABIT[nParmIndex] = ExtractBits(m_nBits4ABIT);
        if (m_pabRiceCodeFlag[nParmIndex]==false) // Adjust for the linear code
            m_pancABIT[nParmIndex] = (m_pancABIT[nParmIndex]==0) ? 0 : m_pancABIT[nParmIndex]+1;
    }
} // ! bUseSegStateCodeParm
```

NOTE: For linear encoding, any non-zero ABIT code is incremented by one.

After the RICE coding flags and the corresponding coding parameters (ABIT) have been unpacked, the remainder of the data is unpacked. This is described in clauses 8.5.2.3 and 8.5.2.5 of the present document.

After unpacking and byte alignment, the position of the bit pointer in bytes shall equal the data section Fsize-1.

8.5.2.3 Entropy Codes Unpacking and Decoding

For each channel, a RICE code flag, a sample bit allocation and residual entropy codes are extracted from the stream. The entropy codes are either binary or RICE codes. Furthermore RICE codes may be either 'Straight' RICE codes (referred to as RICE codes) or 'Hybrid' RICE codes.

The RICE code consists of a variable length unary part (all '0') followed by the stop '1' bit and followed by a binary part of length m_pancABIT[nParmIndex].

The 'Hybrid' RICE coding is employed to guarantee an upper bound (nBits4SamplLoci + m_pancAuxABIT[nParmIndex]) on the length of the unary part of RICE codes. In case of 'Hybrid' RICE codes, some samples are isolated and coded as linear codes of length m_pancAuxABIT[nParmIndex], while the remaining samples are coded as RICE codes. The number and the location index of isolated samples are transmitted in the stream immediately before the residual entropy codes. The residual buffer is initialized to '1' at all locations of isolated samples and to '0' at all remaining locations. This initial state of residual buffer ('0' or '1') is used to indicate the type of extraction that needs to be performed for each residual sample. The two types of extraction are:

- '1' implies extraction of linear code of length m_pancAuxABIT[nParmIndex]; and
- '0' implies extraction of RICE code with coding parameter equal to m_pancABIT[nParmIndex]).

The 'Hybrid' RICE codes can only exist in segments with m_ncSegType = 0. More details about the RICE codes may be found in clause C.8.

The corresponding code is:

```
// Unpack Entropy codes
for (nCh=0; nCh<m_nChSetLLChannel; nCh++){
    pResBuffer = pnInputBuffer[nCh] + m_nSamplFrameIndex;
    // For segment type == 0, distinct coding parameter for each channel
    // For segment type == 1, same coding parameter for all channel
    nParmIndex = (m_ncSegType==0) ? nCh : 0;

    if (m_pabRiceCodeFlag[nParmIndex] == false){
        // =========================   Linear Code  ================================
        if (m_pancABIT0[nParmIndex]>0){
            //=========== Unpack all residuals in one channel of part one of segment 0
            for (n=0; n<m_panSamplPart0[nParmIndex]; n++){
                nTmp = ExtractBits(m_pancABIT0[nParmIndex]);  // Unpack as unsigned
                // Map to signed
                *pResBuffer++ =  (nTmp & 0x1) ? -(nTmp>>1)-1 : nTmp>>1;
            }
        }
        else{
            for (n=0; n<m_panSamplPart0[nParmIndex]; n++)
                *pResBuffer++ =  0;
        }
        if (m_pancABIT[nParmIndex]>0){
    //  Unpack all residuals in one channel of part 2 of segment 0 and all other segments
            for (n=0; n<m_nSmplInSeg-m_panSamplPart0[nParmIndex]; n++){
```

```
                nTmp = ExtractBits(m_pancABIT[nParmIndex]);  // Unpack as unsigned
            // Map to signed
                pResBuffer[n] =  (nTmp & 0x1) ? -(nTmp>>1)-1 : nTmp>>1;
            }
        }
        else{
            for (n=0; n<m_nSmplInSeg-m_panSamplPart0[nParmIndex]; n++)
                pResBuffer[n] =  0;
        }
    }
    else{       // ======================Rice code ================================
        // ============ Unpack all residuals in one channel of part 1 of segment 0
        if (m_pancABIT0[nParmIndex]>0){
            for (n=0; n<m_panSamplPart0[nParmIndex]; n++){
                ExtractUnary(nuTmp); // Extract the unary part
                // Extract the binary part and assemble the unsigned word
                nuTmp = (nuTmp<<m_pancABIT0[nParmIndex]) | ExtractBits(m_pancABIT0[nParmIndex]);
                // Map to signed
                *pResBuffer++ =  (nuTmp & 0x1) ? -((int)(nuTmp>>1))-1 : nuTmp>>1;
            }
        }
        else{
            // For k=0 there is only unary code plus the stop bit
            for (n=0; n<m_panSamplPart0[nParmIndex]; n++){
                ExtractUnary(nuTmp); // Extract the unary part
                // Assemble the Signed word
                // Map to signed
                *pResBuffer++ =  (nuTmp & 0x1) ? -((int)(nuTmp>>1))-1 : nuTmp>>1;
            }
        } // end ncABIT0 condition
        // Set all locations to 0
        memset(pResBuffer,0,sizeof(int)*m_nSmplInSeg-m_panSamplPart0[nParmIndex]);
        // Unpack the number of isolated max samples when Hybrid Rice coding
        if (m_pancAuxABIT[nParmIndex]>0){
            nCountIsMax = ExtractBits(nBits4SamplLoci);
            for (n=0; n<nCountIsMax; n++){
                // Extract the location of isolated max samples and
                // flag the location by 1
                pResBuffer[ExtractBits(nBits4SamplLoci)] = 1;
            }
        }

    // Unpack all residuals in one channel of part 2 of segment 0 and all other segments
        if (m_pancABIT[nParmIndex]>0){
            for (n=0; n<m_nSmplInSeg-m_panSamplPart0[nParmIndex]; n++){
                if (pResBuffer[n]==0){
                    // Pure Rice
                    ExtractUnary(nuTmp); // Extract the unary part
                    // Extract the binary part and assemble the unsigned word
                    nuTmp = (nuTmp<<m_pancABIT[nParmIndex]) | ExtractBits(m_pancABIT[nParmIndex]);
                }
                else
                    // Isolated max binary coded;
                    nuTmp = ExtractBits(m_pancAuxABIT[nParmIndex]);

                // Map to signed
                pResBuffer[n] =  (nuTmp & 0x1) ? -((int)(nuTmp>>1))-1 : nuTmp>>1;
            }
        }
        else{
            // Unpack all residuals in one channel of current segment
            // For k=0 there is only unary code plus the stop bit
            for (n=0; n<m_nSmplInSeg-m_panSamplPart0[nParmIndex]; n++){
                if (pResBuffer[n]==0){
                    // Pure Rice
                    ExtractUnary(nuTmp); // Extract the unary part
                }
                else
                    // Isolated max binary coded;
                    nuTmp = ExtractBits(m_pancAuxABIT[nParmIndex]);
                // Map to signed
                pResBuffer[n] =  (nuTmp & 0x1) ? -((int)(nuTmp>>1))-1 : nuTmp>>1;
            }
        } // end ncABIT condition
    } // end bRiceCodeFlag condition
}
```

If the Rice code flag is false, indicating the use of linear codes and the ABIT allocation is zero, then the residual samples are all zero and the decoder should fill the unpacked array with zeroes. In the case of linear codes that have a non-zero ABIT(s) allocation, the ABIT value is incremented by 1 to accommodate the sign bit of residual that is located at the least significant bit and which is removed prior to sign conversion. The linear-encoded residuals should be extracted using the adjusted bit allocation and converted to a signed value using:

```
*pResBuffer++= (nTmp) & 01) ? -(nTmp>>1) - 1 : (nTmp>>1)
```

Each RICE-encoded residual contains a unary part and a binary part that are extracted separately before combining. For zero bit allocation, the code consists of only the unary part and is extracted by finding the number of zeros preceding the stop bit (1).

The remainder of the RICE code, the binary part, is extracted using ABIT and is converted to a signed value in the same way as the linearly coded residuals. Both unary and binary parts are then combined by shifting the unary part left by (ABIT) places before performing a bit wise logical 'OR' with the binary part.

8.5.2.4 Decimator History Unpacking

Within the MSB data for the first segment of frequency bands 1 and 3, a set of decimator frequency band history coefficients shall be extracted after unpacking the entropy codes.

```
// Unpack decimator history
if (nSeg == 0 && (nFreqBand==1 || nFreqBand==3) )
{
    int nNumBitsForHistSampl;
    nNumBitsForHistSampl = ExtractBits(5)+1; // Unpack m_unNumBitsFBTxHistSamples

for (nCh=0; nCh<m_nChSetLLChannel; nCh++)
    {
        for(n=0;n<7;n++)
            pnDeciHistoryFreqBand[nCh][n] = ExtractBits(nNumBitsForHistSampl);
    }
}
```

The value of nNumBitsForHistSampl will always be 32 bits.

8.5.2.5 LSB Residual Unpacking

If FlagScalableLSBs is set, the LSB part of residuals shall also be extracted. The beginning of the LSB data can be found at:

LSB_Start = Start of current channel set in current frequency band + BandChSetSize[Band][Seg][ChSET] - nLSBFsize[nBand].

If channel sets CRCs are embedded, then *LSB_Start* is reduced by the width of the checksum field.

The parameter nLSBFsize[nBand] is the same size for all segments of frequency band nBand and it is obtained from the transmitted parameter in a channel set's sub header.

The LSB part is linearly coded and should be extracted using the corresponding bit allocation (pnScalableLSB[nBand][ch]) for all segments of frequency band nBand in the current frame. The remaining LSB residuals are simply unpacked as linear codes.

8.5.3 Fixed Coefficient Prediction

The inverse fixed coefficient prediction process, on the decode side, is defined by an order recursive formula for the calculation of k^{th} order residual at sampling instance n:

$$e_k[n] = e_{k+1}[n] + e_k[n-1]$$

where the desired original signal s[n] is given by:

$$s[n] = e_0[n]$$

and where for each k[th] order residual $e_k[-1] = 0$.

In the following example, recursions for the 3rd order fixed coefficient prediction, where the residuals $e_3[n]$ are coded, are transmitted in the stream and unpacked on the decode side:

$$e_2[n] = e_3[n] + e_2[n-1]$$

$$e_1[n] = e_2[n] + e_1[n-1]$$

$$e_0[n] = e_1[n] + e_0[n-1]$$

$$s[n] = e_0[n]$$

The following code demonstrates inverse fixed prediction in the frequency band nBand.

```
for (nCh=0; nCh<m_nNumCh; nCh++){

    // for m_pnFixedPredOrder[nBand][nCh]=0 e0=input sample
       if (m_pnFixedPredOrder[nBand][nCh]>0){
        // Zero out channel working buffer
        memset(&pnWorkResBuffer[0], 0, SIZE_WORK_RES_BUFFER*sizeof(int));
        pnInTemp = pnInputBuffer[nCh];
        for (n=0; n<m_nFrmSize; n++){
            pnBuffTemp = pnWorkResBuffer;
            *pnBuffTemp = pnInTemp[n]; // load residual
            for (nOrd=0; nOrd< m_pnFixedPredOrder[nBand][nCh]; nOrd++){
                // Calculate the ek[n] = ek+1[n] + ek[n-1]
                pnBuffTemp[2] = pnBuffTemp[0] +  pnBuffTemp[1];
                // Save the current residuals for the next sample iteration
                pnBuffTemp[1] = pnBuffTemp[2];
                pnBuffTemp += 2;
            }
            // Save the regenerated sample
            pnInTemp[n] = *pnBuffTemp;
        } // end sample loop in the frame
    } // end if (m_pnFixedPredOrder[nBand][nCh]>0)
} // end channel loop
```

8.5.4 Inverse Adaptive Prediction on the Decode Side

The block diagram in Figure 8-6 depicts the inverse adaptive prediction process on the decoder side.

The first step in performing inverse adaptive prediction is to extract adaptive prediction orders **`PrOr[Ch]`** for each channel **`Ch=1, … NumCh`**. Next, for the channels with **`PrOr[Ch]>0`**, the unsigned version of linear area ratios **`(LAR)`** quantization indices (**`PackLARInd[n] for n=1, … PrOr[Ch]`**) are extracted. For each channel Ch with prediction order **`PrOr[Ch]>0`**, the unsigned **`PackLARInd[n]`** are mapped to the signed values **`QLARInd[n]`** using the following mapping:

$$QLARInd[n] = \begin{cases} PackLARInd[n] >> 1 & \forall\ even\ numbered\ PackLARInd[n] \\ -(PackLARInd[n] >> 1) - 1 & \forall\ odd\ numbered\ PackLARInd[n] \end{cases}$$

$$for\ n = 1, ..., \mathrm{Pr}Or[Ch]$$

where the >> denotes an integer right shift operation.

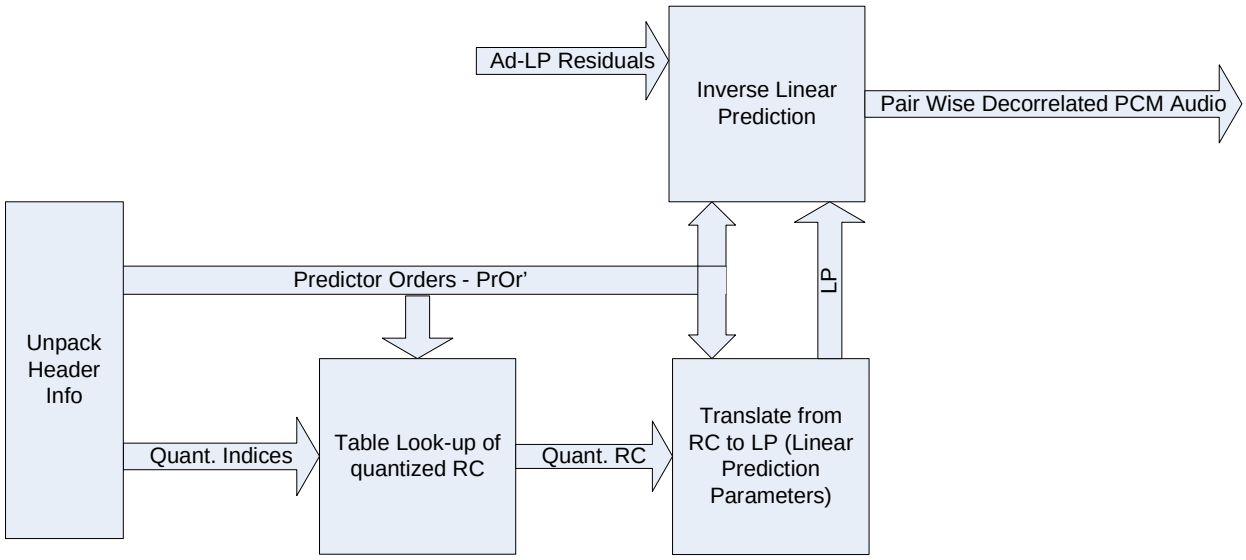


Figure 8-6: Inverse Adaptive Prediction

In the 'Table Look-up of quantized RC' block, an inverse quantization of LAR parameters and a translation to reflection coefficient (RC) parameters is done in a single step using a look-up table **TABLE{}** defined as:

```
TABLE ={0,      3070,  5110,  7140,  9156, 11154, 13132, 15085, 17010,
        18904, 20764, 22588, 24373, 26117, 27818, 29474, 31085, 32648,
        34164, 35631, 37049, 38418, 39738, 41008, 42230, 43404, 44530,
        45609, 46642, 47630, 48575, 49477, 50337, 51157, 51937, 52681,
        53387, 54059, 54697, 55302, 55876, 56421, 56937, 57426, 57888,
        58326, 58741, 59132, 59502, 59852, 60182, 60494, 60789, 61066,
        61328, 61576, 61809, 62029, 62236, 62431, 62615, 62788, 62951,
        63105, 63250, 63386, 63514, 63635, 63749, 63855, 63956, 64051,
        64140, 64224, 64302, 64376, 64446, 64512, 64573, 64631, 64686,
        64737, 64785, 64830, 64873, 64913, 64950, 64986, 65019, 65050,
        65079, 65107, 65133, 65157, 65180, 65202, 65222, 65241, 65259,
        65275, 65291, 65306, 65320, 65333, 65345, 65357, 65368, 65378,
        65387, 65396, 65405, 65413, 65420, 65427, 65434, 65440, 65446,
        65451, 65456, 65461, 65466, 65470, 65474, 65478, 65481, 65485,
        65488, 65491}.
```

The quantized reflection coefficients for each channel **Ch (QRC[n] for n= 1, … PrOr[Ch])** are calculated from the **TABLE{}** and the quantization **LAR** indices **QLARInd[n]**, as:

$$QRC[n] = \begin{cases} TABLE[QLARInd[n]] & \forall QLARInd[n] \geq 0 \\ -TABLE[-QLARInd[n]] & \forall QLARInd[n] < 0 \end{cases}$$

$$for\ n = 1, ..., \Pr Or[Ch]$$

In the next block, for each channel Ch, the quantized RC parameters **QRC_{ord} for *ord* = 1, … PrOr[Ch]** are translated to the quantized linear prediction parameters (**LP_{ord} for *ord* = 1, … PrOr[Ch]**) according to the following algorithm:

$$
\begin{aligned}
&\text{For ord} = 0 \text{ to PrOr - 1 do}\\
&\quad \text{For m} = 1 \text{ to ord do}\\
&\qquad C_{ord+1,\,m} = C_{ord,\,m} + (QRC_{ord+1} \times C_{ord,\,ord+1\text{-}m} + (1 << 15)) >> 16\\
&\quad \text{end}\\
&\quad C_{ord+1,\,ord+1} = QRC_{ord+1}\\
&\text{end}\\
&\text{For ord} = 0 \text{ to PrOr - 1 do}\\
&\quad LP_{ord+1} = C_{PrOr,\,ord+1}\\
&\text{end}
\end{aligned}
$$

Any possibility of saturation of intermediate results is removed on the encode side. Therefore on the decode side, there is no need to perform saturation check after calculation of each $C_{ord+1,\,m}$.

Finally, for each channel with PrOr[Ch]>0, an inverse adaptive linear prediction is performed. Assuming that prediction residuals *e(n)* are previously extracted and entropy decoded, the reconstructed original signals *s(n)* are calculated according to the following equations:

$$
\begin{aligned}
&\overline{s(n)} = \left[\left\{ \sum_{k=1}^{\Pr Or[Ch]} LP_k \times s(n-k) \right\} + (1 << 15) \right] >> 16\\
&Limit \quad \overline{s(n)} \quad to \;\; 24-bit \;\; range \left(-2^{23} \; to \; 2^{23} - 1 \right)\\
&e(n) = s(n) - \overline{s(n)}\\
&for \; n = \Pr Or[Ch] + 1, \ldots \quad NumSamplInFrame
\end{aligned}
$$

Since the sample history is not persistent between the frames, the inverse adaptive prediction shall start from the (PrOr[Ch]+1) sample in the frame.

The first of two related function calls show the conversion of reflection coefficients.

```
// Read the quantized reflection coefficients from the table
for (nOrd = 0; nOrd < nLPCOrder; nOrd++){
    nTmp = pReflCofQind[nOrd];
    nRcq[nOrd] = (nTmp>=0) ? m_pnTanhTbl[nTmp] : -m_pnTanhTbl[-nTmp];
}

// Conversion from reflection coefficients to direct form coefficients
pnRcq = nRcq;
for (nOrd = 1; nOrd <= nLPCOrder; nOrd++){
    pn_F_Coef = naCoeffs;
    pn_B_Coef = naCoeffs + nOrd - 2;
    for (n = 1; n <= (nOrd>>1); n++)
    {
        nTmp = *pn_F_Coef;
        nTmp1 = *pn_B_Coef;
        // No need to check for saturation; prevented in the encoder
        *pn_F_Coef++ = nTmp + (int)((((DTS__int64)(*pnRcq) * nTmp1) + nRound) >> Q_LL_PREDCOEFFS);
        *pn_B_Coef-- = nTmp1 + (int)((((DTS__int64)(*pnRcq) * nTmp) + nRound) >> Q_LL_PREDCOEFFS);
    }
    naCoeffs[nOrd-1] = *pnRcq++;
}

// Reverse the order of coefficients
pn_B_Coef = naCoeffs + nLPCOrder -1;
for (nOrd=0; nOrd<nLPCOrder; nOrd++)
    pnLPCCoef[nOrd] = *pn_B_Coef--;
Inverse adaptive prediction is covered below.
pReflCofQind = m_pnLPCReflCoeffsQInd;
for (nCh=0; nCh<m_nNumCh; nCh++){
    nPredOrd=m_pnAdaptPredOrder[nCh];
    if (nPredOrd>0){
```

```
        // Get the prediction coefficients
        if (!RCInd2Coeffs(nPredOrd, pReflCofQind, pnAdCoeffs))
            return false;
        // Second part of the frame where the residuals need to be
// replaced with actual samples
        nStartResOut = nPredOrd;
        pnInTemp = pnInputBuffer[nCh]+nStartResOut;
        pnFiltBuffer = pnInTemp-nPredOrd;
        for (n=nStartResOut; n<m_nFrmSize; n++){
            ndErr = 0;
            for (nCoefInd=0; nCoefInd<nPredOrd; nCoefInd++){
                ndErr += ((DTS__int64) pnFiltBuffer[nCoefInd])*pnAdCoeffs[nCoefInd];
            }
            // Round and scale the prediction; Coefficients are in Q_LL_PREDCOEFFS
            nxhat = (int) ((ndErr + (1<<(Q_LL_PREDCOEFFS-1)))>>Q_LL_PREDCOEFFS);
            if (nxhat>nHigh)
                nxhat = nHigh;
            else if (nxhat<nLow)
                nxhat = nLow;
            // Calculate the original sample x(n) = err - xhat(n)
            // The saturation check is not needed; it has been taken
// care on the encode side
            nRes = *pnInTemp;
            *pnInTemp++ = nRes - nxhat;

            pnFiltBuffer++;
        } // End sample loop in the frame
    } // end of if(nPredOrd>0)
    pReflCofQind += nPredOrd;
} // end channel loop
```

8.5.5 Inverse Pairwise Channel Decorrelation

When a channel pair has a non-zero pairwise channel decorrelation coefficient, pairwise channel de-correlation (PWChD) shall be applied to the channel pair. The current channel is the basis (*BCh*) and the next channel is the decorrelated channel (*DecCh*). To recover the original channel data (*CorCh*), each of the samples/residuals in the basis channel are scaled by the pairwise channel decorrelation coefficient (α) and then added to the corresponding samples/residuals of the decorrelated channel:

$$CorCh = DecCh + \alpha BCh$$

The original channel order should also be restored before the lossless residuals are combined with the lossy output. (The channel order may have been changed at the encoder to make best use of Pairwise Channel Decorrelation).

Individual channels in a channel set are numbered beginning from 0. The matrix field `OrigChanOrder` indicates the reordering. The following sample code illustrates the decoded buffers being copied and scaled and then reordered.

```
// Inverse channel decorrelation
for (nCh=0; nCh<(m_nNumCh>>1); nCh++){
    ndCoeff = (DTS__int64) pnJRChPairsCoeffs[nCh];
    if (ndCoeff != 0){
        pnB = pnInputBuffer[nCh<<1];
        pnJ = pnInputBuffer[(nCh<<1)+1];
        for (n=0; n<m_nFrmSize; n++)
        *pnJ++ += (int) ( ( ndCoeff*(*pnB++) + nRound)>>nQ );
    }
}

// Reorder channel pointers to the original order
for (nCh=0; nCh<m_nNumCh; nCh++)
    ppSaveInBuffPntrs[nCh] = pnInputBuffer[nCh];
for (nCh=0; nCh<m_nNumCh; nCh++)
    pnInputBuffer[pncOriginalChOrder[nCh]] = ppSaveInBuffPntrs[nCh] ;
```

8.6 Lossless Processes

8.6.1 Assembling the MSB and LSB Parts

Since the residuals in a frequency band may have been encoded using an MSB/LSB split, the MSBs of the residuals will have been scaled as part of the lossless encoding algorithm and the LSBs will appear as binary codes width length equal to NumScalableLSBs[nCh]. The use of this MSB/LSB split is indicated per frequency band and per channel on a frame-by-frame basis in the *bScalableLSB* (for frequency band 0) and *bFlagScalableResExtBand[]* (for each frequency extension band) fields of the header.

When a MSB/LSB split is enabled in a frequency band each channel set of each segment consists of two components namely the MSB and the LSB part, as shown in Figure 8-7. For lossless operation both the MSB and LSB components are required.

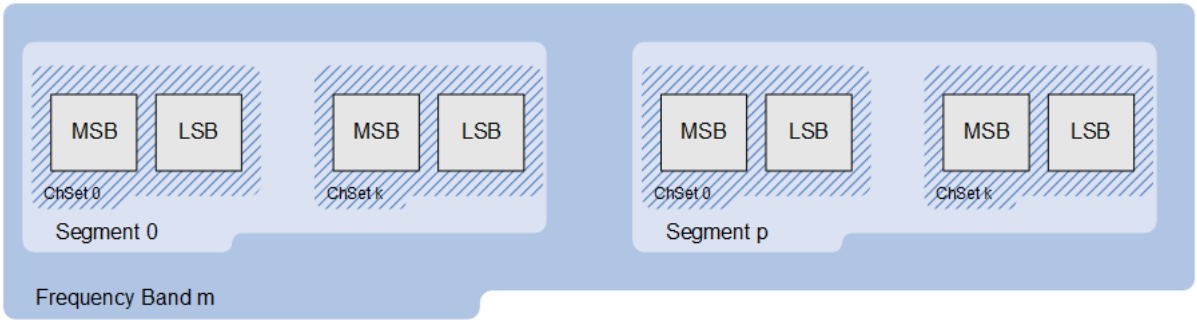


Figure 8-7: Layout of MSB and LSB Data Within One Frequency Band

The MSB component alone represents the most significant portion of the output. The decoder needs to know the width (in bits) of the output and the width of the MSB part in order to properly scale the MSB component before inserting it into the output.

The MSB/LSB split on the encode side may employ either variable or fixed-split point method. The fixed split point method has fixed lengths of the MSB and LSB parts for the entire stream duration and for all channels. The split point is determined by a parameter nuFixedLSBWidth > 0 that represents the length of the LSB part prior to a possible bit width adjustment performed at the authoring stage. Notice that nuFixedLSBWidth > 0 implies the LSB width > 0, which consequently allows for bit-width adjustment in every frame. Although the original MSB/LSB split point is fixed for all channels and all frames, the bit-width adjustment may be variable over the channels and the frames as carried in nBitWidthReduction[nCh]. After performing a variable bit-width adjustment to the stream that has been originally created using the fixed split method (nuFixedLSBWidth > 0) the resulting stream will have variable length LSB parts and consequently nuFixedLSBWidth will be adjusted to be equal to 0.

In contrast, the variable-split point method allows the lengths of the MSB and LSB parts to vary from frame to frame and between the channels. In this case, where parameter nuFixedLSBWidth=0, the split point in each channel nCh, is determined by the NumScalableLSBs[nCh]. This number represents the original width of the LSB part as used during the encode process and prior to a possible bit width adjustment at the authoring stage. Notice, in this case, some frames may have NumScalableLSBs[nCh]=0 and consequently the bit-width adjustment is not possible. This situation may occur for the low levels of input audio signal. However, at the low levels of input audio signal, the variable output bit rate is already low and usually there is no need for bit-width adjustment at the authoring stage.

Both methods allow for dynamic bit-width adjustment. For example, the peak bit rate of some encoded material may exceed the allowed peak bit rate of the medium at very few instances and the peak bit-rate need to be reduced only at those instances; therefore, sacrificing the lossless reconstruction at very few moments of the entire presentation. Unlike the variable split method, the fixed-split point method allows for uniform bit-width adjustment over the entire stream duration and in all channels. As a result, it can guarantee non-varying noise floor in all channels, producing the output audio equivalent to the audio obtained from uniformly reducing the bit-width to the desired number of bits (after applying appropriate dither). The variable-split point method usually gives a better compression performance then the fixed-split point method.

In both, the fixed-split and the variable-split point methods, the NumScalableLSBs[nCh] denotes the length of binary codes that correspond to the samples of the LSB part in channel nCh.

Beginning with an array of assembled MSB sample data and the LSB sample data, the reconstruction of the lossless output (with bit-width equal to nBitResolution) of particular channel nCh, in case when no bit-width adjustment has been performed during the authoring stage, is shown in Figure 8-8. The NumScalableLSBs[nCh] reflects the number of bits that are used to binary code the LSB samples in channel nCh.

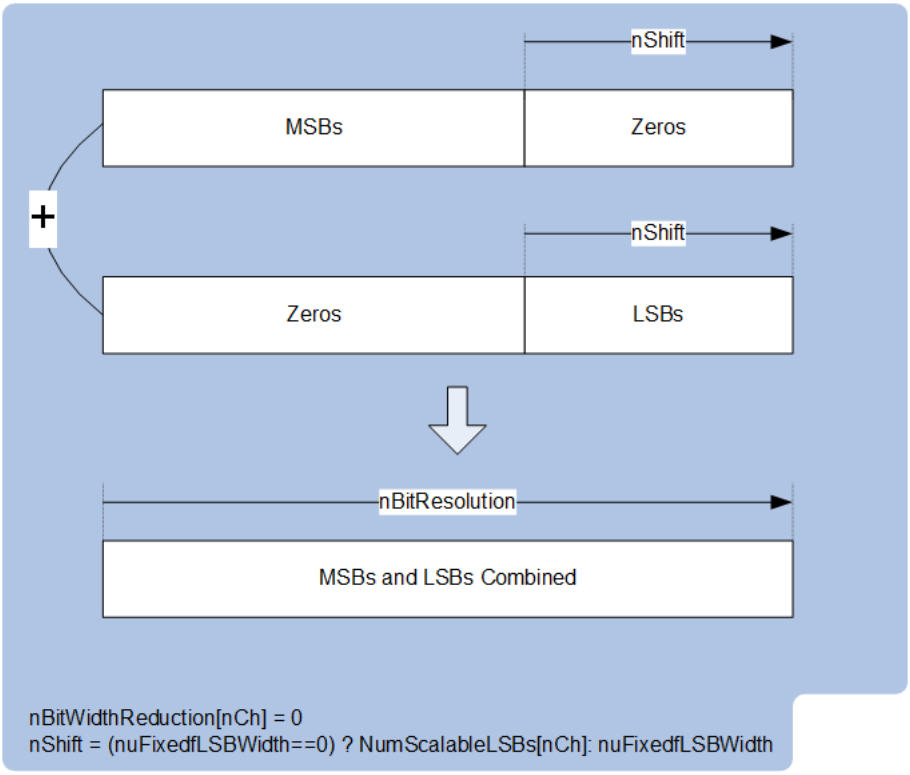


Figure 8-8: Output Word Assembly for Channels Without Bit-width Adjustment (nBitWidthReduction[nCh]=0)

The bit-width management process performed at the authoring stage may reduce the resolution of the samples corresponding to the LSB part of channel nCh by a reduction factor carried in a stream as nBitWidthReduction[nCh]. During this bit-width adjustment process a dither signal is added to the LSB samples. As a consequence the remaining LSB samples after removal of the nBitWidthReduction[nCh] lower significant bits require NumScalableLSBs[nCh] - nBitWidthReduction[nCh] + 1 bits for their transmission. An additional bit corresponding to the + 1 term in the above expression, is a result of adding a dither signal.

Notice that after the bit-width adjustment process, the NumScalableLSBs[nCh], that are transmitted in the stream, are altered such that they still represent the binary code length that is used to code the adjusted samples of the LSB part, i.e.:

$$\text{NumScalableLSBs[nCh]} = \text{NumScalableLSBs[nCh]} - \text{nBitWidthReduction[nCh]} + 1$$

The reconstruction of the lossless output of particular channel nCh, in case when a bit-width adjustment has been performed during the authoring stage, is shown in Figure 8-9. The NumScalableLSBs[nCh] reflects the number of bits that are used to binary code the LSB samples after the bit-width reduction process (denoted as "Remaining LSBs"). The nBitWidthReduction[nCh], indicates the number of lower significant bits that were removed from the LSB samples during the bit-width adjustment process.

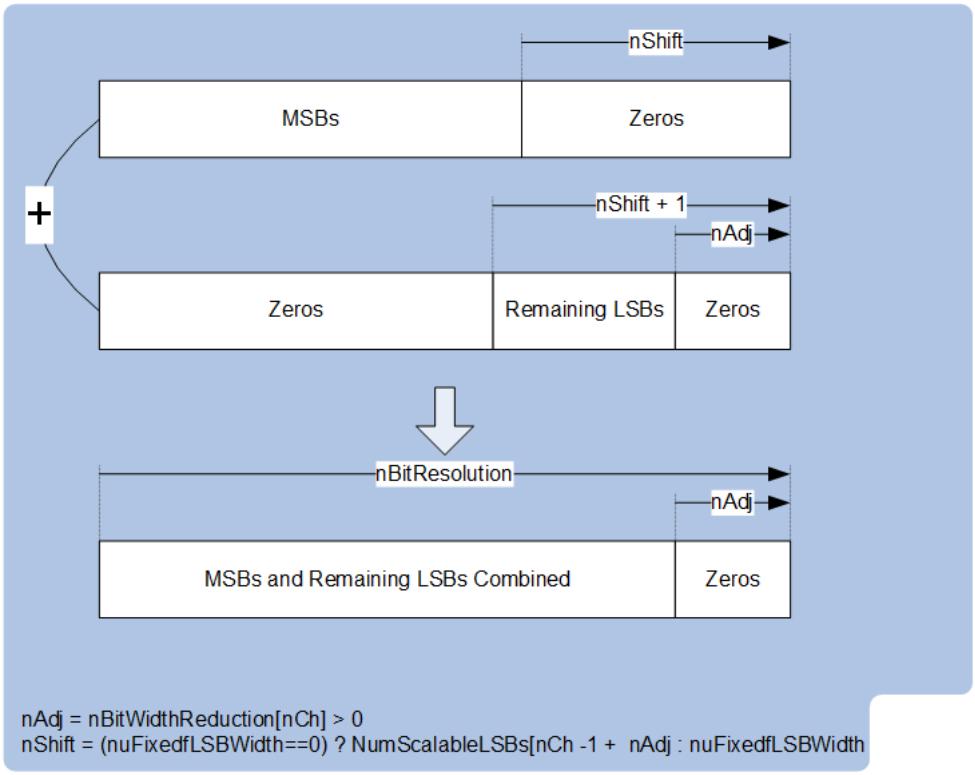


Figure 8-9: Output Word Assembly for Channels with Bit-width Adjustment (nBitWidthReduction[nCh]>0)

The following pseudo-code illustrates assembling MSB and LSB parts:

```
for (nCh=0; nCh<nNumCh; nCh++)
{
nShiftAdj=m_pnBitWidthAdjPerCh[nBandIndex][nCh];
    if (nuFixedLSBWidth==0){
nShift = (nShiftAdj>0 && m_pnScalableLSBs[nBandIndex][nCh]>0) ?
m_pnScalableLSBs[nBandIndex][nCh]-1 : m_pnScalableLSBs[nBandIndex][nCh];
        nShift += nShiftAdj;
    }
else
    {
        nShift = nuFixedLSBWidth;
}

    if (nShift>0){
        pnInSamples = pnInputBuffer[nCh];
        punLSBPart = pnInResAllChLSBs[nCh];
        for (n=0; n<m_nFrmSize; n++){
            nTmp = (*pnInSamples)<<nShift;
            *pnInSamples++ = ( nTmp + ((*punLSBPart++)<<nShiftAdj) );
        }
    }
}
```

To obtain the output words, the resulting samples still need to be shifted left by NumEmptyLSBs places.

8.6.2 Channel Sets Post-Processing

8.6.2.1 Overview of Channel Set Post-Processing

In order to achieve scalability of the stream and the decoding process, all channels presented to the encoder are organized into channel sets. There can be up to 16 channel sets, each with up to 16 channels. Each channel set can be separately extracted from the stream and decoded as desired. The main factors that determine the formation of the channel sets are the following:

1) An intended speaker layout for a channel set, i.e. a 7.1 mix, can be encoded in these 3 channel sets:

   a) Decoding of a channel set 0 produces a stereo downmix.

   b) Decoding and combining of the channel sets 0 and 1 produces a 5.1 downmix.

   c) Decoding and combining of the channel sets 0, 1 and 2 produces the original 7.1 mix.

2) All channels in a channel set always have the same sampling frequency. In a 5.1 stream where the C, L and R channels are at 96 kHz and the Ls, Rs and LFE channels are at 48 kHz, the channels may be organized in two channel sets, where channel set 0 consists of all channels sampled at 96 kHz and channel set 1 consists of all channels sampled at 48 kHz.

3) All channels in a channel set always have the same bit-width. For example, for a 5.1 stream where the L and R channels are 24-bit and the C, Ls, Rs and LFE channels are 16-bit, the channels may be organized in two channel sets, where channel set 0 consists of all channels with 24-bit samples and channel set 1 consists of all channels with 16-bit samples.

4) An intended version of a sound object for a channel set where one version of a sound object can be replaced by another version of the same sound object. For example, there may be a set of channels supporting one version of some sound object (V1_Sound1) and another set of channels supporting a different version of the same sound object (V2_Sound1). The V1_Sound1 and V2_Sound1 channels sets are replacements for each other and they belong to the same replacement group. Only one of the V1_Sound1 or V2_Sound1 channel sets may be designated as the active set. In addition to designating both the V1_Sound1 and V2_Sound1 channel sets as replacement sets, the header will also indicate they are both members of the same replacement group. There is a maximum of three replacement groups.

Primary channels represent the subset of all encoded channels. The primary channels may contain the downmixed version of other non-primary channels. However the primary channels themselves cannot be further used for encoder-embedded downmixing.

The decoding of primary channels is mandatory. An example is a 7.1 stream with an embedded 5.1 downmix that is encoded in two channel sets, where channel set 0 contains a 5.1 downmix and channel set 1 contains two additional surround channels. In this case, the channels that correspond to the 5.1 downmix are primary channels and channel set 0 is denoted as a primary channel set.

The primary channels may be split to a maximum of four primary channel sets with differing sampling frequencies and/or bit widths. This makes it possible to have a scenario where, for the 5.1 stream, the Centre (C), Left (L) and Right (R) channels have a different bit width and/or sampling frequency than the Left Surround (Ls), Right Surround (Rs) and Low Frequency Effects (LFE) channels. In this example, the C, L and R channels are part of one primary channel set and the Ls, Rs and LFE channels are part of another primary channel set.

Although in general, the primary channels in different primary channel sets can have different sampling frequencies, there are restrictions to this rule, which depend on the existence of the lossy core data. When the lossless stream includes the lossy core, all the channels that are coded using both lossy and lossless codecs always have the same sampling frequency. Consequently the primary channels that are coded with both lossy and lossless codecs can only be split into different primary channel sets based on differing bit-widths.

8.6.2.2 Performing and Reversing Channel Set Downmixing

To allow for the scalability of decoder complexity, the audio in the compressed data stream may be organized in multiple channel sets with the intermediate downmix formats (i.e. 10.2 → 7.1 → 5.1 → stereo) embedded in a particular channel set(s).

For example for a 10.2 encoded stream with two embedded downmix configurations (5.1 downmix and stereo downmix), at least three channel sets would be needed, organized such that:

- decoding of a channel set 0 produces a stereo downmix;
- decoding and combining of channel sets 0 and 1 produces a 5.1 downmix; and
- decoding and combining of channel sets 0, 1 and 2 produces the original 10.2 mix.

In the example above, the reconstruction of a 5.1 downmix requires the reversal of the 5.1 → stereo downmix process that has been performed on the encode side. Similarly, the reconstruction of a 10.2 original mix requires the reversal of first 5.1 → stereo and then 10.2 → 5.1 downmix processes that have been performed on the encode side.

Two types of channel-set downmixing are supported: parallel and hierarchical.

8.6.2.3 Parallel Downmix

The parallel type of downmix is performed for non-primary channel sets which have:

- a downmix-embedded enabled flag (bDownmixEmbedded=true);
- a hierarchical-downmix flag disabled (bHierChSet=false); and
- nReplacementSet = 0.

The parallel downmix is performed directly to the primary channel set(s), as shown in Figure 8-10. They are applied in sequential channel-set index order from the first defined parallel (non-primary) channel set to the last defined parallel (non-primary) channel set.

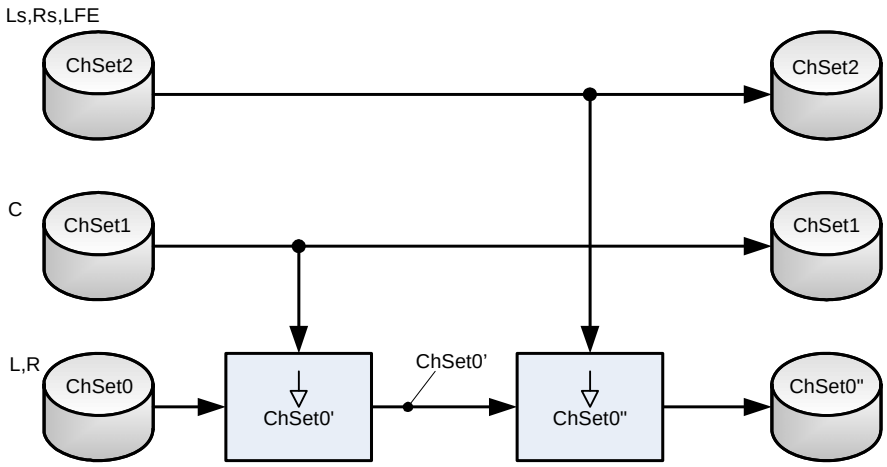


Figure 8-10: Parallel Downmix

On the decode side, the reversal of the parallel downmix is performed in the opposite order - that is, from the last defined parallel (non-primary) channel set to the first defined parallel (non-primary) channel set, as shown in Figure 8-11.

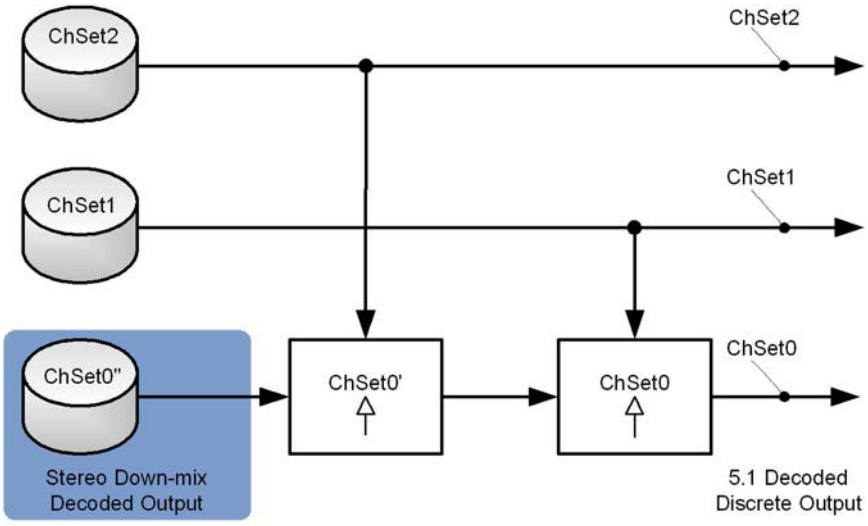


Figure 8-11: Parallel Downmix Reversal

A 5.1 stream organized into three channel sets with the following characteristics provides an example of parallel channel set downmixing and reversal:

1) channel set 0 carries the stereo downmix with 24-bit resolution at 48 kHz;

2) channel set 1 carries C channel with 24-bit resolution at 48 kHz; and

3) channel set 2 carries the Ls, Rs and LFE channels with 16-bit resolution at 48 kHz.

In this example, channel set 0 is a primary channel set. Both channel set 1 and channel set 2 are non-primary channel sets in which bDownmixEmbedded=true, bHierChSet=false and nReplacementSet=0. First channel set 1 is downmixed to channel set 0 and then channel set 2 is downmixed to channel set 0.

On the decode side, the steps are reversed. First channel set 2 is downmixed to channel set 0. Then channel set 1 is downmixed to channel set 0.

8.6.2.4 Hierarchical Downmix

The hierarchical type of downmix, shown in Figure 8-12, is performed for non-primary channel sets that have these three characteristics:

1) a downmix-embedded enabled flag (bDownmixEmbedded=true);

2) a hierarchical downmix flag enabled (bHierChSet=true); and

3) nReplacementSet=0.

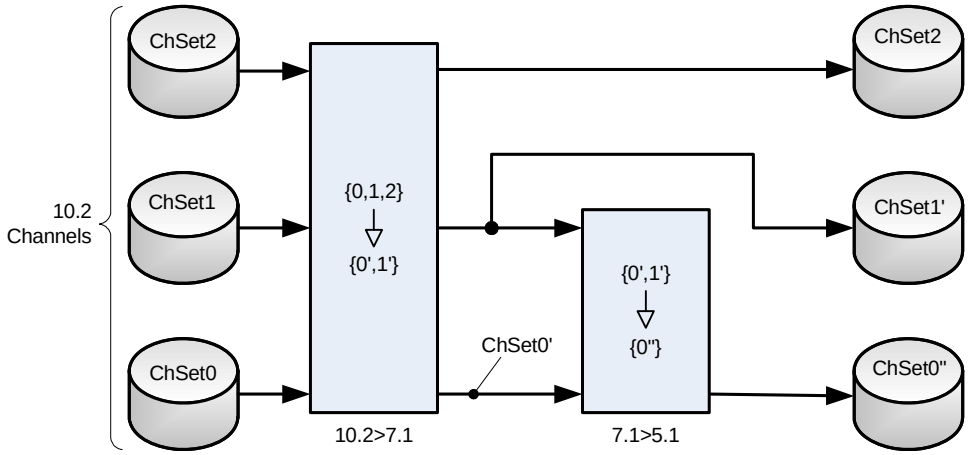


Figure 8-12: Hierarchical Downmix

The hierarchical downmix is performed in a hierarchical manner, moving from the channel set with the highest index to the channel set with the lowest index (primary set(s)). The non-primary channel set may be downmixed in a hierarchical manner, to either:

- the lower-indexed non-primary channel set; or
- the lower-indexed primary channel set.

An example is provided by a 10.2 channel stream with two hierarchically embedded downmix configurations (10.2 → 7.1 → 5.1) organized in three channel sets with the following characteristics:

- decoding of a channel set 0 produces a 5.1 downmix;
- decoding and combining channel sets 0 and 1 produces a 7.1 downmix; and
- decoding and combining channel sets 0, 1 and 2 produces the original 10.2 mix.

Figure 8-12 illustrates the order in which the downmixing occurs. First the 10.2 stream is downmixed to 7.1 (defined in the channel set 2 downmix matrix). Then the 7.1 stream is downmixed to 5.1 (defined in the channel set 1 downmix matrix). If the 7.1 to 5.1 downmix is not defined and a 5.1 output is not requested, no downmixing occurs.

On the decode side (Figure 8-13), the 7.1 → 5.1 process is reversed, by subtracting the channel set 1 contribution (by means of the channel set 1 downmix matrix) from channel set 0. The resulting modified channel set 0 data is to replace the transmitted channel set data. Next the 10.2 to 7.1 process is reversed by subtracting the channel set 2 contribution (by means of the channel set 2 downmix matrix) from both channel set 1 and modified channel set 0.

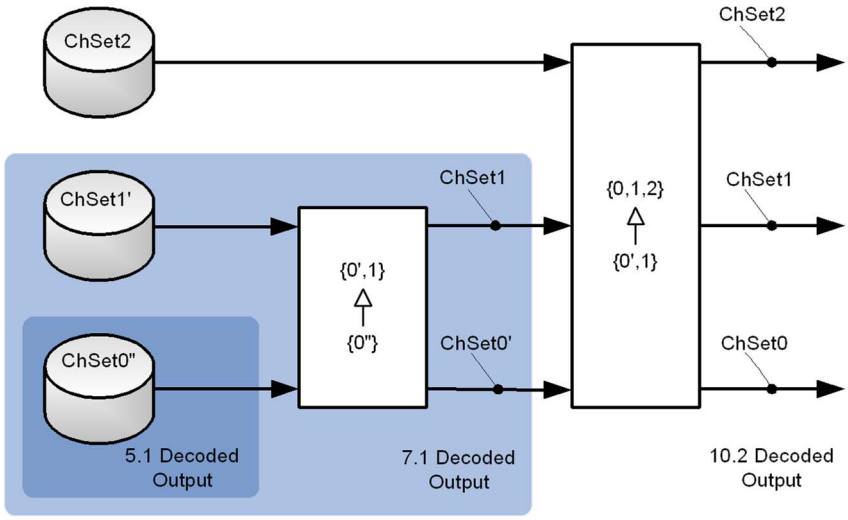


Figure 8-13: Hierarchical Downmix Reversal

9 LBR

9.1 General Information about the LBR Extension

This clause addresses the LBR extension. LBR is a low bit rate coding system used in place of the core and extensions previously described, but operating in the same ecosystem. The LBR payload is used in conjunction with the extension substream header to provide the necessary start-up elements and optional metadata elements required for a complete audio elementary stream.

9.2 The LBR Decoder Environment

9.2.1 General Information About the LBR Decoder

In addition to codec initialization constants carried in, or derived from, information provided in the LBR header, the LBR memory map stores a number of other variable session parameters. Header initialized parameters relate to the audio sampling frequency and the channel map information, which sets up the frame duration, length of a subframe, channel pairing, etc. Other persistent parameters are variables that are usually set to zero at the start of a new decoding session, then are updated by the various algorithms from consecutive coding blocks. Some of these constants and variables are referred to in the decoding block descriptions, so they are described in the following clause.

9.2.2 Persistent Constants and Variables

The following symbols and variable names are used throughout clause 9 describing the LBR extension.

nChannels	Total number of encoded channels.
nPair	Number of channel pairs in the audio stream. nPair is calculated as: $nPair = \left\lfloor \frac{nChannels+1}{2} \right\rfloor$
nSampleRate	The audio sampling frequency. (see Table 9-3).
FreqRange	A coded parameter derived from nSampleRate (see Table 9-4).
nFrameDuration	The number of rendered audio samples per channel per audio frame (Table 9-11).
nBitsLeft	The number of bits yet to be extracted from the chunk being processed (initialized by the chunk length parameter then decremented as bits are parsed).
Qlevels	Quantization levels, this array is configured in the High Resolution chunk.
SecChPres	This is a flag indicating that a secondary channel is present. This flag is set in DecodeTS() if a secondary channel can be decoded.
TSCodingMethod	TSCodingMethod is a flag that indicates whether time samples are packed using Huffman codes or packed directly. This parameter is configured in DecodeTS().

9.3 LBR Extension Substream Header

The syntax of the LBR audio header is described in Table 9-1.

Table 9-1: LBR Extension Substream Header Structure

Syntax	Size (Bits)
`// Extract sync word 0x0a801921`	
`SYNCEXTLBR = ExtractBits (32)`	32
`// Extract LBR header type:`	
`ucFmtInfoCode = ExtractBits(8)`	8
`if (ucFmtInfoCode == 2)`	
`{  // LBR decoder initialization data follows:`	8
`nLBRSampleRateCode = ExtractBits(8)`	
`usLBRSpkrMask = ExtractBits(16)`	16
`nLBRversion = ExtractBits(16)`	16
`nLBRCompressedFlags = ExtractBits(8)`	8
`nLBRBitRateMSnybbles = ExtractBits(8)`	8
`nLBROriginalBitRate_LSW = ExtractBits(16)`	16
`nLBRScaledBitRate_LSW = ExtractBits(16)`	16
`} // end if (ucFmtInfoCode == 2)`	
`else if (ucFmtInfoCode != 1)`	
`{`	
`// unknown ucFmtInfoCode: resync to next SYNCEXTSSH`	
`}`	
`// LBR compressed audio data follows`	Extends to next SYNCEXTSSH sync word, as determined by nuExtSSFsize in Extension SubstreamHeader

SYNCEXTLBR (extension substream sync word for LBR)

The extension substream has a DWORD-aligned synchronization word DTS_SYNCWORD_LBR with the hexadecimal value of 0x0a801921. By default the sync word (DWORD) will only occur on a DWORD boundary (4 Bytes).

ucFmtInfoCode

This unsigned 8-bit value declares the LBR header type. Currently two LBR header types are defined (LBR_HDRCODE_SYNC_ONLY and LBR_HDRCODE_DECODERINIT). If an undefined header type is encountered, the decoder should resync to the next DTS-HD Extension Substream sync word. The first header type that a decoder should handle is LBR_HDRCODE_DECODERINIT because this header is necessary to initialize the LBR decoder with bitrate, samplerate, channel count and flags.

Table 9-2: ucFmtInfoCode values

LBR header type	Value	Description
LBR_HDRCODE_SYNC_ONLY	1	Header consists only of LBR sync word and this ucFmtInfoCode byte; thus raw LBR audio data follows.
LBR_HDRCODE_DECODERINIT	2	Header consists of LBR sync word, this ucFmtInfoCode byte and LBR decoder initialization data. Raw LBR audio data will follow the LBR decoder initialization data.
Reserved	All undeclared values are reserved for future use.	

nLBRSampleRateCode (sample rate of LBR audio samples)

The sample rate of the LBR decoded audio. This 8 bit value is a lookup into a sample rate table and defines the sample frequency of the decoded audio samples. nSampleRate, as shown in Table 9-3.

Table 9-3: nLBRSampleRateCode Sample Rate Decoding

nLBRSampleRateCode	nSampleRate
0	8 000
1	16 000
2	32 000
3	reserved
4	reserved
5	22 050
6	44 100
7	reserved
8	reserved
9	reserved
10	12 000
11	24 000
12	48 000
13	reserved
14	reserved
15	reserved

If a valid frame is encountered in which the nLBRSampleRateCode sample rate differs from the nLBRSampleRateCode sample rate of the previous valid frame, it would indicate that a new stream has been encountered. In this case, the existing LBR decoder would need to be re-initialized with the new header values (including the new nLBRSampleRateCode sample rate) and decoding input and output buffers re-allocated to accommodate the new sample rate.

The decoder parameter FreqRange is set according to nSampleRateCode according to Table 9-4.

Table 9-4: FreqRange

Range of Source Sampling Frequency	FreqRange
nSampleRate < 16 000	0
16 000 ≤ nSampleRate < 32 000	1
32 000 ≤ nSampleRate < 50 000	2

usLBRSpkrMask (LBR speaker mask)

This 16-bit value little-endian value describes the speaker mask for the LBR audio asset. usLBRSpkrMask follows the same convention as the nuSpkrActivityMask defined in Table 7-10.

nLBRversion (LBR bitstream version)

This 16-bit field represents the LBR bitstream version number, (distinct from the overall DTS-HD version). The version is represented in little-endian format as:

`0xMMmm ,`

where:

- 'MM' is the major version number expressed in the high order byte; and
- 'mm' is the minor revision number expressed in the low order byte.

A decoder should reject LBR substreams whose LBR bitstream major version number does not match the decoder's LBR bitstream major version number.

nLBRCompressedFlags (flags for LBR decoder initialization)

This 8-bit field is a compressed version of the LBR initialization flags. Table 9-5 shows the bit positions of the flags within nLBRCompressedFlags.

Table 9-5: Parameter nLBRCompressedFlags

Bit 7 (MSB)	6	5	4	3	2	1	0
Reserved	Multi-channel downmix: 0=none 1=present	Stereo downmix: 0=none 1=present	Bandlimit Flag MSB	Bandlimit Flag	Bandlimit Flag LSB	LFE Channel: 0=none 1=present	Sample Size: 0=16 bits 1=24 bits

The flag descriptions may be found in Table 9-6.

Table 9-6: LBRFlags from bLBRCompressedFlags

nLBRCompressedFlags	Corresponding LBR Flags	Description
0b00000001	LBR_FLAG_24_BIT_SAMPLES	If 1: the input/output PCM audio samples are 24 bits in length. Otherwise the samples are 16 bits in length.
0b00000010	LBR_FLAG_USE_LFE	If 1: LFE channel is present.
0b00011100	LBR_FLAG_BANDLMT_MASK	Mask to isolate flags which describe bandlimit factors, which enhance audio quality with certain sample rate/bitrate combinations.
0b00100000	LBR_FLAG_STEREO_DOWNMIX	If 1: Stereo downmix is present within bitstream.
0b01000000	LBR_FLAG_MULTICHANNEL_DOWNMIX	If 1: Multi-channel downmix is present within bitstream.

The bandlimit flag bits indicate upsampling in the decoder is necessary to restore the original sampling frequency. The supported resampling ratios are shown in Table 9-7.

Table 9-7: LBR band limit flags from nLBRCompressed Flags

nLBRCompressedFlags	Corresponding LBR Flag values	Description
0b00000100	LBR_FLAG_BANDLMT_FACTOR_2_3	Limit bandwidth to 2/3 of original bandwidth.
0b00001000	LBR_FLAG_BANDLMT_FACTOR_1_2	Limit bandwidth to1/2 of original bandwidth.
0b00001100	LBR_FLAG_BANDLMT_FACTOR_1_3	Limit bandwidth to 1/3 of original bandwidth.
0b00010000	LBR_FLAG_BANDLMT_FACTOR_1_4	Limit bandwidth to 1/4 of original bandwidth.
0b00011000	LBR_FLAG_BANDLMT_FACTOR_1_8	Limit bandwidth to 1/8 of original bandwidth.
0b00010100	LBR_FLAG_BANDLMT_FACTOR_NONE	Do not change bandwidth.
all unused values	Reserved for future use	

nLBRBitRateMSnybbles (most-significant nibbles of LBR stream original and scaled bitrates)

The LBR bitstream header carries two bitrates:

- the original bitrate;
- the scaled bitrate which may be used when scaling has been performed on the LBR audio frame after it was encoded.

Each bitrate is expressed as a 20-bit value. The 8-bit nLBRBitRateMSnybbles bitstream value carries the most-significant 4-bit nibble of the original bitrate and the most-significant 4-bit nibble of the scaled bitrate.

The LBR original bitrate is determined by the following equation:

nOriginalBitRate = nLBROriginalBitRate_LSW | ((nLBRBitRateMSnybbles & 0x0F) << 16).

The LBR scaled bitrate is determined by the following equation:

nScaledBitRate = nLBRScaledBitRate_LSW | ((nLBRBitRateMSnybbles & 0xF0) << 12).

nLBROriginalBitRate_LSW (least significant word of LBR original bitrate)

The LBR original bitrate is expressed as a 20-bit value.

nLBROriginalBitRate-LSW is a little-endian 16 bit field which contains the least-significant 16 bits of the encoded LBR stream's original bitrate.

The LBR original bitrate is determined by the following equation:

nOriginalBitRate = nLBROriginalBitRate_LSW | ((nLBRBitRateMSnybbles & 0x0F) << 16).

nLBRScaledBitRate_LSW (least significant word of LBR scaled bitrate)

The LBR scaled bitrate is expressed as a 20-bit value.

nLBRScaledBitRate-LSW is a little-endian 16 bit field which contains the least-significant 16 bits of the encoded LBR stream's scaled bitrate.

The LBR scaled bitrate is determined by the following equation:

nScaledBitRate = nLBRScaledBitRate_LSW | ((nLBRBitRateMSnybbles & 0xF0) << 12).

9.4 LBR Audio Data Organization

9.4.1 General Information About LBR Structure

The LBR audio payload is organized in a modular form referred to here as a "chunk". All audio data is contained within a chunk or series of chunks.

9.4.2 Chunks

9.4.2.1 General Information About LBR Chunks

The chunk is the basic building block of the LBR bitstream. An LBR chunk is composed of three fields: the chunk ID, chunk length and chunk data.

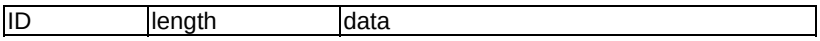


Figure 9-1: Basic Chunk Description

Organization of the LBR stream into chunks allows for:

- simplification of adding new bitstream features;
- forward compatibility. As new techniques and enhancements are developed, old decoders will still be able to play the frame portions that they are aware of;
- scalable bitstream. Simplified post-encode bit-rate scaling of the LBR bitstream;
- separation of coding components to implement robust interleaving and unequal error protection.

Chunks may be nested. The higher level chunk is referred to as the "parent" of the chunks it contains. Likewise the chunks contained within a parent chunk are referred to as a "child chunk" of the parent. The LBR Frame Chunk, for example, is the highest level chunk in the LBR bitstream.

9.4.2.2 Chunk ID

The `chunkID` is used to identify the type of data stored in the chunk. The chunk ID is an 8-bit value providing a total of 256 unique chunk IDs. An extension ID is defined to allow for an additional 256 chunk types after the initial 256 have been used. The most significant bit of the chunk ID is used to indicate whether the chunk is a short chunk (maximum data size of 255 bytes) or a standard chunk (maximum data size of 65 535 bytes). A zero bit indicates a short chunk while a one bit indicates a standard chunk.

Table 9-8 lists the binary values of the Chunk IDs that have been defined.

Table 9-8: Chunk ID Table

ID Value	Name	Description
x000 0000	nullID	Null Chunk
x000 0001	padID	Program Associated Data Chunk
x000 0100	LBR_ID	LBR Frame Chunk with checksum
x000 0110	LBR_ID_no_checksum	LBR Frame Chunk without checksum
x000 1010	LFE_ADPCM_ID	LFE Chunk
x000 1011	EmbLevels_ID	Embedded Channel Sets Chunk
x000 1100	WmarkID1	Reserved
x000 1101	WmarkID2	Reserved
x000 1110	scalefactor_ID	Tonal Scale Factors Chunk
x001 0000	tonal_ID	Tonal Data Chunk (combined)
x001 0001	tonalGroup1_ID	Tonal Data Chunk Group 1
x001 0010	tonalGroup2_ID	Tonal Data Chunk Group 2
x001 0011	tonalGroup3_ID	Tonal Data Chunk Group 3
x001 0100	tonalGroup4_ID	Tonal Data Chunk Group 4
x001 0101	tonalGroup5_ID	Tonal Data Chunk Group 5
x001 0110	tnlScf_ID	Tonal Data Chunk (combined, scalefactors used)
x001 0111	tnlScf_Group1_ID	Tonal Data Chunk Group 1 (scalefactors used)
x001 1000	tnlScf_Group2_ID	Tonal Data Chunk Group 2 (scalefactors used)
x001 1001	tnlScf_Group3_ID	Tonal Data Chunk Group 3 (scalefactors used)
x001 1010	tnlScf_Group4_ID	Tonal Data Chunk Group 4 (scalefactors used)
x001 1011	tnlScf_Group5_ID	Tonal Data Chunk Group 5 (scalefactors used)
x011 pppp	grid1ID	Residual Data Chunk. 1st scalefactors grid
x100 pppp	hiGridsID	Residual Data Chunk. High resolution scalefactor grids
x101 pppp	tsmp1ID	Residual Data Chunk. Timesamples, 1st part
x110 pppp	tsmp2ID	Residual Data Chunk. Timesamples, 2nd part
x111 1111	extensionID	Extension ID

The most significant bit of the chunk ID (`x`) is used to indicate whether the chunk is a short chunk (0) or a standard (1) chunk. 'pppp' in residual chunk ID values is used to indicate channel pair number (0 to 15).

9.4.2.3 Extended ID Chunks

To allow for an additional 256 chunk IDs, an extended ID mechanism is defined. When **`chunkID`** is equal to `extensionID`, the 8 bits following the Length field are the extended ID.

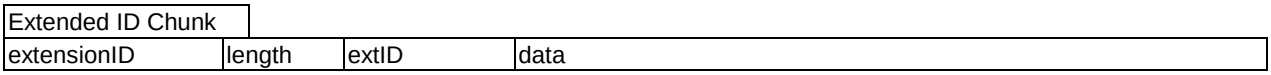


Figure 9-2: Extended ID Chunk

9.4.2.4 Chunk Length

The `chunkLength` indicates the length of the data portion of the chunk in bytes. The chunk length is 8-bits for short chunks and 16-bits for standard chunks, thus the maximum data size of a short and standard chunk is 255 and 65 535 bytes respectively.

Table 9-9: ChunkLengthInfo

Syntax	Size (Bits)

ChunkLengthInfo()
{
chunkID = ExtractBits(8)
if((chunkID & 0x80) == 0){
chunkLength = ExtractBits(8)
return (1)
}
else {
chunkLength = ExtractBits(16)
return (2)
}
}
	
	8
	
	8
	
	16

If **chunkID** byte has the most-significant bit clear, the following 8 bits are an unsigned integer representing the chunk length. If **chunkID** byte has the most-significant bit set, the following 16 bits (most significant byte first) are an unsigned integer representing the chunk length.

9.4.2.5 Data

The data field contains the data of the chunk. The format of the data is specific to the type of chunk.

9.4.2.6 Checksum Verification and Descrambling

The checksum is the 16-bit sum of all the bytes in the frame except for two checksum bytes. The LBR Frame Chunk does not always carry all types of tabulated chunks. Some chunks may be missing due to bitstream fitting operations, transmission channel losses, or may be intentionally not coded during encoding. The decoder still should be able to recover remaining information from chunks present.

If chunkID equals LBR_CID, a checksum has been computed and is stored in the bitstream in the 2 bytes following the chunk length. chunkHeaderBytelen will be 2 if ChunkLengthInfo() is 8 bits long, or 3 if ChunkLengthInfo() is 16 bits long.

Table 9-10: ChecksumVerify

SYNTAX	Size (Bits)

ChecksumVerify (chunk)
{
nChecksum = 0
nHeaderLength = ChunkLengthInfo()+1
nChunkLength -= 2
// Get checksum stored in the bit stream (MSBF)
nStoredChecksum = ExtractBits(16)
// Calculate the checksum on the header
for (i = nHeaderLength-1; i >= 0; i--)
nChecksum += chunk[i]
// Calculate the checksum on the frame data
// Note: the stored checksum is not included in the calcuation
i = nHeaderLength + 2
count = nHeaderLength + nDataLength
for (; i < count; i++){
nChecksum += chunk[i]
bCheckSumSuccess = (nChecksum == nStoredChecksum)
return bCheckSumSuccess ? SUCCESS : FAILURE
}
	
	Table 9-9
	
	16

9.5 LBR Frame Chunk

The LBR bitstream is organized in audio frames, each representing a uniform time period. The number of audio samples generated from each frame is dependent on the sample rate of the decoded audio. The LBR Frame Chunk is the parent chunk to contain all coded audio and side information chunks for a given audio frame.

The coded audio chunks within the frame are organized in a sequence. A chunk may be either an elemental component of the encoded audio or a collection of these elementary component chunks.

Tonal chunks and residual chunks comprise the two main component chunk types. If a Low Frequency Effects (LFE) channel exists, it is coded using a special LFE chunk type.

Both the tonal and residual chunks have multiple resolutions of components and the residual chunks are further coded in a multi-resolution grid structure.

A typical LBR frame is shown in Figure 9-3.

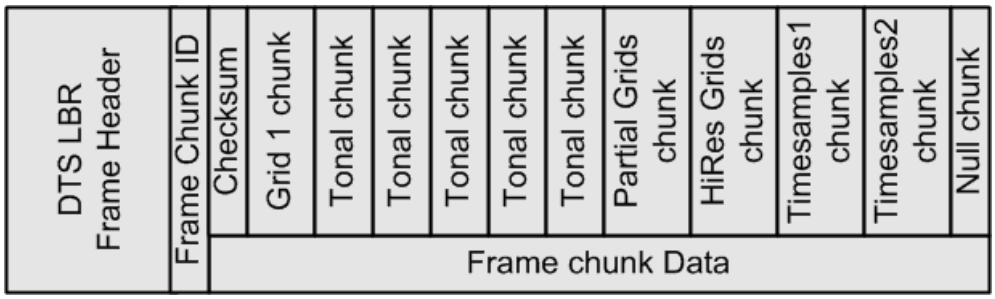


Figure 9-3: Example LBR Frame

9.6 LBR Decoding

9.6.1 Overview of LBR Decoding

The decoding process is illustrated in Figure 9-4. The input data for the decoding process consists of the input bitstream containing packed data frames and minimal side-band information consisting of:

- Sample rate (Hz)
- Number of channels
- Bit rate of the input bitstream (bits/sec)

Decoding can be done on a frame-by-frame basis or at a smaller subframe granularity. The entire decoding process can be divided into number of separate decoding sub-processes:

- Bitstream parsing
- Tonal part decoding
- Residual decoding
- LFE channel decoding (if LFE is present)

The output of each decoding sub-process (other than bitstream parsing) is a sequence of time domain samples corresponding to the data in the current frame (or subframe). These output time-samples should be added together (no shift or alignment required) to obtain the complete decoded signal data. Note that unpacking each packed frame provides samples for the current frame plus additional samples which should be added to the results of the next frame that is decoded.

Decoding of LFE channels is independent of the other channels.

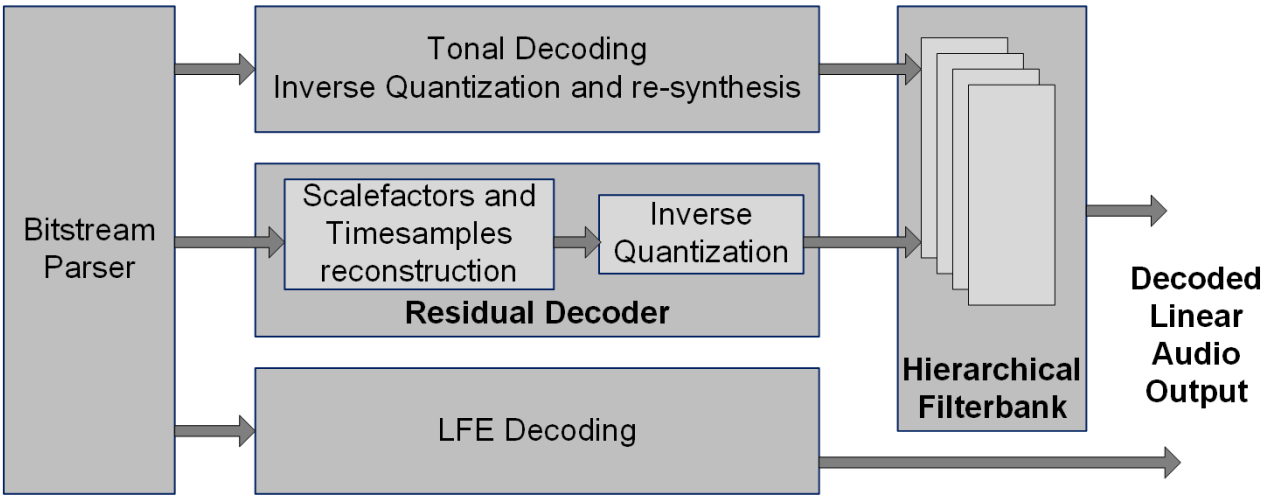


Figure 9-4: LBR Decoder Overview

The encoded data stream is composed of a sequence of individual frames. The number of samples resulting from decoding a frame depends on the sample rate of the audio signal and nominal values are shown in Table 9-11.

Table 9-11: Sample Rate to Frame Size Relationship

Sample Rate	Number of input samples
< 16 kHz	1 024 samples
≥ 16 kHz but < 32 kHz	2 048 samples
≥ 32 kHz but < 50 kHz	4 096 samples

The LBR decodes each input frame as 16 smaller subframes.

Table 9-12: Decode Frame

Syntax	Size (Bits)

`DecodeFrame(){`
`  if (chunkID == LBR_ID){`
`  if(FAILURE == ChecksumVerify(chunk))`
`  return`
`  }`
`  for (sf=0; sf < 16; sf++)`
`  DecodeSubFrame(sf)`
`}`
	Table 9-10
	Table 9-13

Table 9-13: Decode SubFrame

Syntax	Size (Bits)
`DecodeSubFrame(nSubFrameCount){`	
`  InitializeParameters()`	Table 9-20
`  if (bLBRCompressedFlags && (LBR_FLAG_STEREO_DOWNMIX || LBR_FLAG_MULTICHANNEL_DOWNMIX))`	
`  ECSChunk()`	Table 9-38
`  if (bLBRCompressedFlags && (LBR_FLAG_USE_LFE))`	
`  LFEChunk()`	Table 9-36
`  if (nSubFrameCount == 0)`	
`  ResidualChunksPart1()`	Table 9-21
`  // Grid1Chunk`	
`  // HiResGridChunk`	
`  // TimeSamples1Chunk`	
`  // TimeSamples2Chunk`	
`  // tonal vs residual shift is 11 subframes`	
`  if (nSubFrameCount == 11){`	
`  ScaleFactorChunk()`	Table 9-14
`  TonalChunk()`	Table 9-15
`  }`	
`  for (ch=0, ch < nPair, ch+=2){`	
`  ResidualChunksPart2(ch, nSubFrameCount)`	Table 9-32
`  // complete decoding of residual subbands`	
`  }`	
`  padChunk()`	Table 9-39
`  nullChunk()`	Table 9-40
`}`	

9.6.2 Tonal Decoding

9.6.2.1 Overview of Tonal Decoding

The tonal chunks are used to store the tonal coding information of the LBR algorithm. Either a separate chunk is used for each different transform size (also called 'group'), or all transform sizes together with tonal scale factors are collected into one chunk.

The information for tonal decoding process consists of base functions divided into groups by length. Base functions are spread by time in the frame and only some of them have non-zero values in a given subframe. There are 5 tonal groups in total. The length of the base functions in each of the groups, measured in terms of subframes, is as follows:

- 1st group (`nGroup=0`) length is 2 subframes;
- 2nd group (`nGroup=1`) length is 4 subframes;
- 3rd group (`nGroup=2`) length is 8 subframes;
- 4th group (`nGroup=3`) length is 16 subframes;
- 5th group (`nGroup=4`) length is 32 subframes.

The difference values obtained from the bitstream for the secondary audio channels (for stereo and multi-channel signals) should be converted to absolute values. The amplitude component on the secondary channel is always lower than the amplitude of the primary channel. Quantized amplitudes should be converted to linear values using the quantized amplitude to linear amplitude conversion table in clause 9.9.1.

Tonal decoding steps are as follows:

- Initialize frequency domain subframe with zero values.
- Unpack tonal scale factors.
- Unpack tonal components and adjust each with the corresponding scale factor.

- Synthesize and add required portions of 1st-5th group of base functions into same subframe.
- Convert frequency domain subframe data into the time domain using an inverse MDCT followed by windowing. This step can be combined with residual reconstruction in the second stage of the hybrid filterbank.

Note that the base function synthesis operations (step 4) can be performed in any order.

9.6.2.2 Tonal Scale Factors Chunk

9.6.2.2.1 Tonal Scale Factor Chunk Syntax

The Scale Factors Chunk is used to store scale factors for tonal components.

Table 9-14: ScalefactorsChunk()

Syntax	Size (Bits)
`ScaleFactorsChunk()`	
`{`	
`    if (chunkID != scalefactor_ID) {`	8
`        return`	
`    ChunkLengthInfo()`	Table 9-9
`    for (scfBand=0; scfBand < 6; scfBand++)`	
`        nScaleFactor[scfBand] = ExtractBits(6)`	6
`    ByteAlign()`	0..7
`}`	

9.6.2.2.2 Tonal scale factor processing

At the encoder, the minimum amplitude out of all components within a correspondent frequency range is selected as its scale factor, so each component belonging to this frequency range is then adjusted with the scale factor before packing.

At the decoder, it is not necessary to de-quantize the tonal scale factors; instead, the unpacked value should be added to the quantized amplitude of all tonal components within the correspondent frequency range before de-quantization. In the case when more than one channel having a component at the same frequency, amplitude of the channel with highest level should be treated as described above and the quantized amplitude difference of all secondary channels (channels at lower level) should be subtracted from treated maximum amplitude before quantization, see clause 9.6.2.3 for details.

Correspondence between frequency and scalefactor index is described in Table 9-4 (see 'FreqToSf').

9.6.2.3 Tonal Chunks

9.6.2.3.1 About Tonal Chunks

Two different versions of chunks exist, depending on whether scale factors are used. Those versions are identical from the bitstream point of view, except for the `chunkID` field. All tonal chunks have the syntax shown below.

When a separate chunk is used for each different transform size, or group:

Table 9-15: TonalChunk()

Syntax	Size (Bits)
`TonalChunk()`	
`{`	
`    if (chunkID == tonalGroupX_ID || chunkID == tnlScf_GroupX_ID) {`	
`    // where X is a digit between 1 and 5 which represents transform size`	8
`        ChunkLengthInfo()`	Table 9-9
`        DecodeTonal(X-1)`	Table 9-17
`        ByteAlign()`	0..7
`}   }`	

When all transform sizes together with tonal scale factors collected into one chunk:

Table 9-16: TonalChunk() (with scalefactors)

Syntax	Size (Bits)
`TonalChunk()`	
`{`	
`    if (chunkID != tnlScf_ID) {`	8
`        return`	
`    ChunkLengthInfo()`	Table 9-9
`    for (i=0, i < 5; ++i)`	
`    switch (nChunkID)`	
`    {`	
`        default:`	
`        case tnlScf_Group5_ID:`	
`        case tonalGroup5_ID:`	
`            nGroup = 0`	
`        break;`	
`        case tnlScf_Group4_ID:`	
`        case tonalGroup4_ID:`	
`            nGroup = 1`	
`        break;`	
`        case tnlScf_Group3_ID:`	
`        case tonalGroup3_ID:`	
`            nGroup = 2`	
`        break;`	
`        case tnlScf_Group2_ID:`	
`        case tonalGroup2_ID:`	
`            nGroup = 3`	
`        break;`	
`        case tnlScf_Group1_ID:`	
`        case tonalGroup1_ID:`	
`            nGroup = 4`	
`        break;`	
`    }`	
`    DecodeTonal(nGroup)`	Table 9-17
`    ByteAlign()`	0..7
`}`	

Code common to both types of tonal chunk types:

Table 9-17: Decode Tonal

Syntax	Size (Bits)
`DecodeTonal(nGroup)`	
`{`	
`   nSubFrame = 1`	
`   nFrequency = 1`	
`   nFrequencyDiff = 0`	
`   RemainingTonesForThisGroup = 1`	
`   iterations = 1`	
`   while(RemainingTonesForThisGroup){`	
`      while (iterations){`	
`         nFrequencyDiff = getVariableParam(prsDist[4-nGroup])`	Table 9-18
`         if (nFrequencyDiff>>2 > 0){`	
`            bitlength = nFrequencyDiff >> 2`	
`            nFrequencyDiff = ExtractBits(bitlength) + fstAmp[nFrequencyDiff]`	bitlength
`         }`	
`         else{`	
`            nFrequencyDiff = fstAmp[nFrequencyDiff]`	
`         }`	
`         if (nFrequencyDiff > 1) {`	
`            iterations = 0`	
`            break`	
`      }`	
`      nFrequency = 1`	
`      if (nFrequencyDiff = 0)`	
`         nSubFrame += 1`	
`      else`	
`         nSubFrame += 8`	

Syntax	Size (Bits)
`if(nSubFrame > (1 << nGroup))`	
`  RemainingTonesForThisGroup = 0`	
`}`	
`nFrequency += nFrequencyDiff – 2`	
`if ( nChannels > 1)`	
`  mainChIdx = ExtractBits(BitsForChNum[nChannels] – 1)`	variable
`else`	
`  mainChIdx = 0`	
`tonalAmplitudeMain = getVariableParam(tnlScf) + nScalefactors[FreqToSf[nFrequency >> (7-nGroup)]]`	Table 9-18
`tonalPhaseMain = ExtractBits(3)`	3
`for (ch = 1; ch < nChannels; ch++) {`	
`  if (ch != mainChIdx) {`	
`  chPresence = ExtractBits(1)`	1
`  if (chPresence){`	
`  nAmplitudeDiff = getVariableParam(dAmp)`	Table 9-18
`  nPhaseDiff = getVariableParam(dPh)`	Table 9-18
`  tonalAmplitudeSecondaryDiff = TonalAmplitudeMain – nAmplitudeDiff`	
`  tonalPhaseSecondaryDiff = tonalPhaseMain – nPhaseDiff`	
`} } }`	
`nFrequency++`	
`} }`	

where:

tables for prsDist[n], tnlScf, dAmp and dPh are defined in clause 9.9.10,

```
fstAmp [44] =
{
   0,    1,    2,    4,
   4,    6,    8,    10,
   12,   16,   20,   24,
   28,   36,   44,   52,
   60,   76,   92,   108,
   124,  156,  188,  220,
   252,  316,  380,  444,
   508,  636,  764,  892,
   1020, 1276, 1532, 1788,
   2044, 2556, 3068, 3580,
   4092, 5116, 6140, 7164
}

FreqToSf [32] =
{
   0,
   1,
   2, 2,
   3, 3, 3, 3,
   4, 4, 4, 4, 4, 4, 4, 4,
   5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5
}


BitsForChNum[10] =
{           // channel number
            1, // 0
            1, // 1
            2, // 2
            2, // 3
            3, // 4
            3, // 5
            3, // 6
            3, // 7
            4, // 8
            4  // 9
}
```

Table 9-18: getVariableParam()

syntax	Size (Bits)

getVariableParam (table)
{
index = 1
nbits = 0
while(table[index].A != 0xFF){
Nvalue = table[index].ExtractBits(1) //0 == A, 1 == B
index += Nvalue
}
Nvalue = table[index].B
if (nValue = 0){
nbits = ExtractBits(3)+4
return (ExtractBits(nbits))
}
else
return (Nvalue -1)
}
	1
	
	3
	nbits

`GetVariableParam()` is used by both Tonal and Residual functions to fetch variable bit length parameters from the bitstream. The table entries each have a sub-index referring to column A (0) or column B (1) and result in a table walk or "index hopping" to arrive at the correct offset in statistically the fewest number of bits.

The tables passed into this function are all in clause 9.9.10. The resulting index is used to locate a base value in `FstAmp[]` which also determines the number of additional bits required to extract from the bitstream to code the residual value.

9.6.2.3.2 Tonal components processing

In the bit stream, the components are packed in ascending frequency order and each active bin position may contain component from one or more channels. In the multi-channel case, the channel with the highest amplitude would appear first, coded using absolute amplitude (with scale factor adjustment) and phase. All secondary channels at lower levels are then difference-encoded against the maximum channel.

Bitstream parsing provides the following values (see clause 9.4):

- quantized amplitude (`LogAmplitude`);
- quantized phase (`Phase`);
- spectral line number (`Freq`);
- position of base function in the frame;
- per-channel 'presence' of the component - the bit map of the channels where component has non-zero amplitude.

After the scale factor/maximum channel adjustment, quantized amplitudes should be converted to linear values using the quantized amplitude to linear conversion table in clause 9.9.1.

Frequency domain subframe data consists of a number of MDCT spectral lines. The number of lines (N) used depends on the sample rate according to Table 9-19.

Table 9-19: Subframe Resolution

Sample rate	Number of spectral lines in subframe
< 16 kHz	64
≥ 16 kHz but < 32 kHz	128
≥ 32 kHz but < 50 kHz	256

NOTE: This frequency domain data results in 2 × N time domain samples. The first half of these samples should be added to the last half from the previously decoded subframe (which is automatically done by hybrid filterbank). The resulting N samples are the output of the tonal decoding process.

9.6.2.3.3 Base-functions synthesis

The Base function is a finite-length time-domain function used in tonal encoding and decoding. The base function is described by the formula:

$$F(t;A,l,f,\varphi) = A \cdot \frac{1-\cos(\frac{2\pi}{l}\cdot t)}{2} \cdot \sin(\frac{2\pi}{l}\cdot f \cdot t + \varphi) \text{ where } t \in [0,l)$$

$$F(t;A,l,f,\varphi) = 0;$$

where:

A amplitude;

t time variable ($t \in N$);

φ phase;

l function length, power of 2 (=128..8192);

f frequency $f \in [1, \frac{l}{2})$

Real functions used in the algorithm are approximations of this function and described by fast-synthesis algorithm below. This formula is also used to create the table 'wavSynEnvelope[]' which is used to shape the tonal components (base functions) in tonal coding.

Tonal base functions are spread over a number of subframes. The output of synthesizing these base functions is applied to all subframes where the synthesized base function has non-zero values. The synthesis is done by updating Amplitude and Phase values according to the frequency and the length of the base function. A number of neighbour spectral lines are also modified by 'CorrCf[][]' to reduce distortions produced by this synthesis method. Pseudo-code below illustrates the process:

```
ph0_shift[8]  = {-32, +96, -96, +32, +96, -32, +32, -96};
xFreq = Freq >> (nGroup + 1);
F_dlt = Freq & (1<<(nGroup+1)-1) << (4-nGroup);
PhaseRotation = 256 - ((F_dlt + (xFreq&1)*32)*4);
CurPhase = 128 - 64*Phase - ((PhaseRotation << nGroup+1)-PhaseRotation) +
           ph0_shift[(xFreq&3)*2 + ((Freq &1))] + (xFreq>>1);
for (index = 0; index < Length; index = index+1) {
  CurAmplitude = Amplitude * wavSynEnvelope[index];
  WavSyn[index][xFreq-5] += CurAmplitude*CorrCf[F_dlt][0] * sin(2*Pi*CurPhase/256 - 5*Pi/2);
  WavSyn[index][xFreq-4] += CurAmplitude*CorrCf[F_dlt][1] * sin(2*Pi*CurPhase/256 - 4*Pi/2);
  WavSyn[index][xFreq-3] += CurAmplitude*CorrCf[F_dlt][2] * sin(2*Pi*CurPhase/256 - 3*Pi/2);
  WavSyn[index][xFreq-2] += CurAmplitude*CorrCf[F_dlt][3] * sin(2*Pi*CurPhase/256 - 2*Pi/2);
  WavSyn[index][xFreq-1] += CurAmplitude*CorrCf[F_dlt][4] * sin(2*Pi*CurPhase/256 - 1*Pi/2);
  WavSyn[index][xFreq]   += CurAmplitude*CorrCf[F_dlt][5] * sin(2*Pi*CurPhase/256);
  WavSyn[index][xFreq+1] += CurAmplitude*CorrCf[F_dlt][6] * sin(2*Pi*CurPhase/256 + 1*Pi/2);
  WavSyn[index][xFreq+2] += CurAmplitude*CorrCf[F_dlt][7] * sin(2*Pi*CurPhase/256 + 2*Pi/2);
  WavSyn[index][xFreq+3] += CurAmplitude*CorrCf[F_dlt][8] * sin(2*Pi*CurPhase/256 + 3*Pi/2);
  WavSyn[index][xFreq+4] += CurAmplitude*CorrCf[F_dlt][9] * sin(2*Pi*CurPhase/256 + 4*Pi/2);
  WavSyn[index][xFreq+5] += CurAmplitude*CorrCf[F_dlt][10]* sin(2*Pi*CurPhase/256 + 5*Pi/2);
  CurPhase += PhaseRotation;
}
```

where:

WavSyn	spectral line of MDCT
Freq	values obtained from the bitstream (see Table 9-17)
Amplitude	linear amplitude of base function, derived from *LogAmplitude* by using the table in clause 9.9.1
wavsynEnvelope[index]	envelope values obtained from the table in clause 9.9.2
nGroup	group number (0 for shortest base functions, 4 for longest)
Length	length of base function (`Length = 1 << nGroup+1`)

CorrCf[F_dlt][ ]	correction coefficient given by the table in clause 9.9.3 (corresponding table should be used for each given group)
F_dlt	difference between original frequency and center frequency of the spectral line used for synthesis

As the synthesis is done on a subframe basis only one iteration of the cycle should be done and `CurPhase` and *CurAmplitude* should be stored for synthesizing the next subframe.

If the spectral line number referred to by the algorithm appears to be negative (this occurs when *xFreq* is 0 to 4 and the calculation refers to the terms *[xFreq-5]* to *[xFreq-1]*), a mirroring effect takes place. The mirrored spectral line (the number of this spectral line is the absolute value of the negative spectral line number) should be corrected using the negative of the complex tabulated coefficient.

9.6.3 Residual Decoding

9.6.3.1 About Residual Decoding

Residual decoding is a scalable scheme. Each residual frame produces the same number of samples as one LBR frame.

9.6.3.2 Residual Decoding Overview

At the encoder, the residual samples of each primary channel are processed by filters producing 32 uniform frequency subbands, each containing 128 time samples. From the subband samples, scale factors are calculated to normalize the samples before quantization.

There are potentially 4 different residual coding chunks, in each LBR frame:

- Grid1 chunk - contains low resolution grid of scale factors.
- High resolution grids chunks (two types) - contains high-time resolution grid of scale factors and high-frequency resolution grid of scale factors (Grid2 and Grid3).
- Timesamples chunks - time sample information for each of the filterbank subbands.

Residual decoding involves the following steps:

1) Unpack and decode Grid1, Grid2, Grid3 scale factors, then use them to reconstruct the high resolution (Hi Res) scale factor grid.
2) Unpack LPC coefficients and use them with the prediction error unpacked in step 3 to synthesize the subband samples.
3) Unpack LPC errors and time samples using quantizers specified by the quantization profile.
4) Rescale with corresponding Hi Res scale factors.
5) Reconstruct residual time samples with inverse filterbank.

These steps are further described in the following clauses.

9.6.3.3 Unpacking and Decoding Residuals

9.6.3.3.1 Decoding Residuals Syntax

Table 9-20: Residual parameter initialization

```
Syntax
InitializeParameters()
{
    FrequencyRange = SAMPLERATE_TO_FREQRANGE(nSampleRate)
    nTotalSubbands = 8 << (FrequencyRange)
    nBaseBitRate = (500+250 * nChannels)
    if (nBitRate >= 176 * nBaseBitRate)
        nResProfile = 2
    else if (nBitRate >= 100 * nBaseBitRate)
        nResProfile = 1
    else
        nResProfile = 0
    G3AvgOnlyStartSB = nTotalSubbands * 1000 * Profiles[nResProfile].nAvgGrid3Frequency /
(nSampleRate/2)
    if (G3AvgOnlyStartSB > nTotalSubbands)
        G3AvgOnlyStartSB = nTotalSubbands
    MinMonoSubband = nTotalSubbands * 1000 * Profiles[nResProfile].nMinMonoFrequency /
(nSampleRate/2)
    if (MinMonoSubband > nTotalSubbands)
        MinMonoSubband = nTotalSubbands
    MaxMonoSubband = nTotalSubbands * 1000 * 14 / (nSampleRate/2)
    if (MaxMonoSubband > nTotalSubbands)
        MaxMonoSubband = nTotalSubbands
    SecChPres = 0
   TSCodingMethod = 0
}
```

where:

```
   Profiles[] = {nMinMonoFrequency,nAvgGrid3Frequency}
Profiles[3] =
   {
      { 2, 16 },
      { 2, 18 },
      { 2, 24 }
   }
```

SAMPLERATE_TO_FREQRANGE() is defined by Table 9-4 and other persistent parameters are defined in clause 9.2.2.

Table 9-21: Residual Chunks Part 1

syntax	Size (Bits)
ResidualChunksPart1(){	
for (ch=0, ch < nPair, ch+=2){	
// the last pair of channel is mono	
// when the number of fullband channel is odd	
if ((ch+2) = nChannels){	
oneChPair = TRUE	
}else{	
oneChPair = FALSE	
}	
Grid1Chunk(ch, oneChPair)	Table 9-23
HiResGridChunk(ch, oneChPair)	Table 9-24
TimeSamples1Chunk(ch, oneChPair)	Table 9-22
TimeSamples2Chunk(ch, oneChPair)	Table 9-25
} }	

Table 9-22: TimeSamples 1 Chunk

syntax	Size (Bits)
TimeSamples1Chunk(ch, oneChPair){	
DecodeLPC(2, 3)	Table 9-31
DecodeTS(ch, 2, 4, 0, oneChPair)	Table 9-28
DecodeGrid2(0, 1)	
DecodeTS(ch, 4, 6, 0, oneChPair)	Table 9-28
}	

Table 9-23: Grid1 Chunk

syntax	
Grid1Chunk(ch,oneChPair){	
// decode scale factors nSubbandScfGrid1	
nTotalSubbandsG1 = ScalefactorToGrid1[nTotalSubbands-1] + 1	
for (nSubband = 2; nSubband < nTotalSubbandsG1; nSubband++){	
DecodeScaleFactors(nSubbandScfGrid1[ch][nSubband][])	Table 9-29
}	
// decode average values for 3rd grid	
if (nTotalSubbands > 4)	
nTotalSubbandsG3 = nTotalSubbands-4 + 1	
else	
nTotalSubbandsG3 = 1	
for (nSubband = 0; nSubband < nTotalSubbandsG3; nSubband++){	
nSubbandAvgGrid3[ch][nSubband] = getVariableParam(avgG3)	Table 9-18
nSubbandAvgGrid3[ch][nSubband] -= 16	
if (oneChPair=FALSE){ // "stereo" pair	
if ((nSubband+4) >= 2){	
// copy primary to secondary	
nSubbandAvgGrid3[ch+1][nSubband] = nSubbandAvgGrid3[ch][nSubband];	
}	
else{ // read data from bitstream for the 2 first subbands	
nSubbandAvgGrid3[ch+1][nSubband] = getVariableParam(avgG3);	Table 9-18
nSubbandAvgGrid3[ch+1][nSubband] -= 16;	
} } }	
// get stereo image for partial mono mode	
if (nBitsLeft >= 8){	
if (oneChPair=FALSE){	
nPartialStereoSubband = 0	
nMin[0] = ExtractBits(4);	4
nMin[1] = ExtractBits(4);	4
for (nSubband = 2, nSubband < nTotalSubbands; nSubband += 4){	
for (nCh = ch; nCh <= (ch+1); nCh++){	
for (nSubSample = 3; nSubSample >=0; nSubSample--){	
nSubbandPartStereo[nCh][nPartialStereoSubband][nSubSample] = getVariableParam(stGrid) + nMin[nCh-ch]	Table 9-18
} }	
nPartialStereoSubband++	
} } }	
// get low-resolution spatial information	
for (nCh = 2; nCh < nChannels; nCh++){	
nMin[0] = ExtractBits(4)	4
nPartialStereoSubband = 0	
for (nSubband = 0, nSubband < nTotalSubbands; nSubband += 4){	
for (nSubSample = 3; nSubSample >=0; nSubSample--){	
nSubbandPartStereo[nCh][nPartialStereoSubband][nSubSample] = getVariableParam(stGrid) + nMin[nCh-ch]	Table 9-18
}	
nPartialStereoSubband++	
} } }	

where

`stGrid` and `avgG3` are defined in clause 9.9.10.

Table 9-24: High Resolution Grid Chunk

syntax	Size (Bits)
HiResGridChunk(ch, oneChPair) {	
// quantizer profile is read from bitstream	
nQuantizerProfile = ExtractBits(8)	8
// calculate quantization levels for each subband	
OL = (nQuantizerProfile & 0x38) >> 3 // OL: overall level	
ST = (nQuantizerProfile & 0xC0) >> 6 // ST: steepness	
maxSB = nQuantizerProfile & 0x07	
// calculate levels according to a formula	
for (sb=0; sb<nTotalSubbands; ++sb){	
F = sb*nSampleRate/nTotalSubbands	
A = 18000/(12*F/1000+100+40*ST) + 20*OL	
// translate dB into quantization level indices	
if (A <= 95)	
QLevels[sb] = 1 // 1.0 bits	
else if (A <= 140)	
QLevels[sb] = 2 // 1.6 bits	
else if (A <= 180)	
QLevels[sb] = 3 // 2.4 bits	
else if (A <= 230)	
QLevels[sb] = 4 // 3.0 bits	
else	
QLevels[sb] = 5 // 4.0 bits	
}	
// reorder quantization levels for lower subbands according to maxSB	
QLevels[maxSB] = QLevels[0]	
// get LPC for the first two subbands	
DecodeLPC(0, 2)	Table 9-31
// get time-samples for the first two subbands of main channel	
DecodeTS(ch, 0, 2, 0, oneChPair)	Table 9-28
// get the first two bands of the first grid from bitstream	
for (nSubband = 0; nSubband < 2; nSubband++)	
{	
for (nCh = ch; nCh <= (ch+1); nCh++){	
DecodeScaleFactors(nSubbandScfGrid1[nCh][nSubband][])	Table 9-29
} } }	

Table 9-25: TimeSamples 2 Chunk

syntax	Size (Bits)
TimeSamples2Chunk(ch, oneChPair){	
DecodeGrid2(1, 4)	
DecodeTS(ch, 6, nMaxMonoSubband, 0, oneChPair)	Table 9-28
if (oneChPair = FALSE){	
DecodeGrid1(ch)	
DecodeGrid2(ch, 0, 4)	
}	
DecodeTS(ch, nMinMonoSubband, nTotalSubbands, 1, oneChPair)	Table 9-28
}	

Table 9-26: Decode Grid1

syntax	Size (Bits)
DecodeGrid1(ch){	
nTotalSubbandsG1 = ScalefactorToGrid1[nTotalSubbands-1] + 1	
for (nSubband = 2; nSubband < nTotalSubbandsG1; nSubband++){	
DecodeScaleFactors(nSubbandScfGrid1[ch+1][nSubband][])	Table 9-29
} }	

where:

```
ScalefactorToGrid1[64] =
{
 0,1,2,3,4,4,5,5,6,6,6,6,7,7,7,7,8,8,8,8,8,8,9,9,9,9,10,10,10,10,10,10,
 11,11,11,11,11,11,11,11,11,11,11,11,12,12,12,12,12,12,12,12,12,12,12,
 12,12,12,12,12,12,12,12
}
```

Table 9-27: Decode Grid2

syntax	Size (Bits)
DecodeGrid2(ch, startSB, endSB){	
nTotalSubbands = ScalefactorToGrid2[nTotalSubbands-1] + 1	
if (endSB > nTotalSubbands)	
endSB = nTotalSubbands	
for (nSubband = startSB; nSubband < endSB; nSubband++){	
for (nCh = ch; nCh <= (ch+1); nCh++){	
for (i = 63; i >= 0; i -= 8){	
nValue = ExtractBits(1)	1
if (nValue){	
for (j = 7; j >= 0; j--){	
nValue = ExtractBits(5)	5
r = Grid2Codes[nValue]	
r &= 31	
if(r<31)	
nSubbandScfGrid2[nCh][nSubband][j] = r	
if(ExtractBits(1)){	1
nSubbandScfGrid2[nCh][nSubband][j] = 6	
break	
if(ExtractBits(1)){	1
nSubbandScfGrid2[nCh][nSubband][j] = 7	
break	
}	
if(ExtractBits(1)){	1
nSubbandScfGrid2[nCh][nSubband][j] = 8	
break	
}	
if(ExtractBits(1)){	1
nSubbandScfGrid2[nCh][nSubband][j] = 9	
break	
}	
if(ExtractBits(3) = 3){	3
nSubbandScfGrid2[nCh][nSubband][j] = 0xa	
break	
}	
if(ExtractBits(3) = 3){	3
nSubbandScfGrid2[nCh][nSubband][j] = 0xb	
break	
}	
if(ExtractBits(7) = 2){	7
nSubbandScfGrid2[nCh][nSubband][j] = 0xc	
break	
}	
if(ExtractBits(7) = 6){	7
nSubbandScfGrid2[nCh][nSubband][j] = 0x10	
break	
}	
if(ExtractBits(15) = 5){	15
nSubbandScfGrid2[nCh][nSubband][j] = 0xe	
break	
}	
if(ExtractBits(15) = 9){	15
nSubbandScfGrid2[nCh][nSubband][j] = 0x11	
break	
}	
if(ExtractBits(31) = 1){	31
nSubbandScfGrid2[nCh][nSubband][j] = 0x12	
break	

syntax	Size (Bits)

}
if(ExtractBits(31) = 17){
nSubbandScfGrid2[nCh][nSubband][j] = 0xd
break
}
if(ExtractBits(31) = 0xd){
nSubbandScfGrid2[nCh][nSubband][j] = 0xf
break
}
// rare value case
nBits = ExtractBits(3) + 4
r = ExtractBits(nBits)
if(r > 56)
nSubbandScfGrid2[nCh][nSubband][j] = 0
nSubbandScfGrid2[nCh][nSubband][j] = r
}
}else{
for (j = 7; j >= 0; j--){
nSubbandScfGrid2[nCh][nSubband][j] = 0
} } } } } }
	31
	31
	3
	nbits

where:

```
ScalefactorToGrid2[64] =
{
   0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 1, 1, 1, 1, 1, 1,
   1, 1, 2, 2, 2, 2, 2, 2,
   2, 2, 2, 2, 2, 2, 2, 2,
   3, 3, 3, 3, 3, 3, 3, 3,
   3, 3, 3, 3, 3, 3, 3, 3,
   3, 3, 3, 3, 3, 3, 3, 3,
   3, 3, 3, 3, 3, 3, 3, 3
}


Grid2Codes[32] =
{
   66,   99,   65,   64,   66,   132,  65,   64,   66,   99,   65,   64,   66,   165,  65,   64,
   66,   99,   65,   64,   66,   132,  65,   64,   66,   99,   65,   64,   66,   191,  65,   64
}
```

Table 9-28: DecodeTS

syntax	Size (Bits)

DecodeTS(ch,startSB,endSB,channelFlag, oneChPair){
for (nSubbandIndex = startSB; nSubbandIndex < endSB ; nSubbandIndex++){
nSbReordered = 0
if (nSubbandIndex > 6){
if ((channelFlag=0) | (nSubbandIndex >= nMaxMonoSubband))
nSbReordered = ExtractBits(5)
}else{
nSbReordered = nSubbandIndex
}
// get grid3 scf
if (nSubbandIndex = 12){
if (G3AvgOnlyStartSB > 4)
nEndSubbandG3 = G3AvgOnlyStartSB-4
else
nEndSubbandG3 = 0
for (nSubbandG3 = 0; nSubbandG3 < nEndSubbandG3; nSubbandG3++)
{
for (nCh = ch; nCh <= (ch+1); nCh++)
nSubbandScfGrid3[nCh][nSubbandIndex][nSubbandG3] = DecodeGrid3()
}
}else if ((nSbReordered >= 4) & (nSubbandIndex < 12)){
if (nSbReordered > 4)
	5
	Table 9-30

syntax	Size (Bits)
nSubbandG3 = nSbReordered-4	
else	
nSubbandG3 = 0	
for (nCh = ch; nCh <= (ch+1); nCh++){	
nSubbandScfGrid3[nCh][nSubbandIndex][nSubbandG3] = DecodeGrid3()	Table 9-30
} }	
// stereo matrix	
if (oneChPair=FALSE){	
if (nBitsLeft >= 20)	
nSubbandMidSide[nSbReordered] = ExtractBits(8)	8
if ((channelFlag = 1) & (nSbReordered >= nMinMonoSubband))	
nLRMS = ExtractBits(8)	8
}	
nStartChannel = ch	
nEndChannel = ch+2	
if ((nSbReordered >= nMinMonoSubband) & (nSubbandIndex < nMaxMonoSubband))	
{	
if (channelFlag = 0)	
nEndChannel = ch+1	
else	
nStartChannel = ch+1	
}	
for (nCh = nStartChannel; nCh < nEndChannel; nCh++){	
overCoded = 0	
sbCodingFetched = 0	
for (nSubFrame = 0; nSubFrame < 4; nSubFrame++){	
// timesample coding method is fetched from the bitstream only once	
if (!sbCodingFetched){	
TSCodingMethod[nCh][nSbReordered][nSubFrame] = ExtractBits(1)	1
sbCodingFetched = 1	
}	
switch (QLevels[nSubbandIndex]) {	
case 1:	
nBitsToExtract = min(nBitsLeft, 32)	
break	
case 2:	
if (TSCodingMethod[nCh][nSbReordered][nSubFrame] == 0){	
// in the first method, 5 samples are combined to form an	
// 8-bit number (3^5 = 243 which fits into 8 bits)	
if ((32-overCoded) % 5)	
nBitsToExtract = min(nBitsLeft, 8 * ((32-overCoded) / 5 + 1))	
else	
nBitsToExtract = min(nBitsLeft, 8 * ((32-overCoded) / 5))	
overCoded = max(0, 5*nBitsToExtract/8 - (32-overCoded))	
}else{	
i = 0	
j = nBitPosition	
nValue = ExtractBits(j)	variable
while (j > 0){	
tmv = (nValue & 1) + 1	
nValue = nValue >> tmv	
j -= tmv	
i++	
}	
nBits = nBitsLeft	
for (i=0;i < 16 & (nBits >= 16); nBits -= 16){	
nValue = ExtractBits(1)	1
nValue |= ExtractBits(1) << 8	1
if (j < 0)	
nValue = nValue >> 1	
j += 16	
while (j > 0){	
tmv = (nValue & 1) + 1	
nValue = nValue >> tmv	
j -= tmv	
i++	
} }	
nBitsLeft = nBits	

syntax	Size (Bits)
`if (j < 0)`	
`ExtractBits(1)`	1
`for (i=0;i < 32;i++){`	
`if (nBitsLeft >= 10){`	
`for (j = 0; j < 5; j++){`	
`if (i + j >= 32)`	
`break`	
`if (ExtractBits(1))`	1
`ExtractBits(1)`	1
`} }`	
`i += j`	
`} }`	
`break`	
`case 3:`	
`if ( (32-overCoded) % 3 )`	
`nBitsToExtract = min(nBitsLeft, 7 * (32-overCoded) / 3 + 1)`	
`else`	
`nBitsToExtract = min(nBitsLeft, 7 * (32-overCoded) / 3 )`	
`overCoded = max(0, 3*nBitsToExtract/7 - (32-overCoded))`	
`break`	
`case 4:`	
`for (i = 0; i < 32; i++)`	
`nCodeVal[ExtractBits(8) & 63]`	8
`break`	
`case 5:`	
`nBitsToExtract = min(nBitsLeft, 4 * 32)`	
`break`	
`}`	
`if (nBitsToExtract){`	
`if (nBitsToExtract >= 16){`	
`nBitsLeft -= ((nBitsToExtract >> 3) - 2) << 3`	
`ExtractBits(16)`	16
`ExtractBits( nBitsToExtract & 7 )`	variable
`}else{`	
`ExtractBits(nBitsToExtract)`	variable
`} } } }`	
`if (!oneChPair & (nSubbandIndex >= nMinMonoSubband) & (nBitsLeft >= 20))`	
`SecChPres = 1`	
`} }`	

where:

```
nCodeVal[64]=
{
 1, 2, 4, 3, 5, 2, 0, 3, 1, 2, 4, 3, 6, 2, 0, 3, 1, 2, 4, 3, 5, 2, 0, 3, 1, 2, 4, 3, 7, 2, 0, 3,
 1, 2, 4, 3, 5, 2, 0, 3, 1, 2, 4, 3, 6, 2, 0, 3, 1, 2, 4, 3, 5, 2, 0, 3, 1, 2, 4, 3, 4, 2, 0, 3
}
```

Table 9-29: Decode Scalefactors

syntax	Size (Bits)
`DecodeScaleFactors(*nSubbandScf){`	
`subframe = 0`	
`nPrevious = getVariableParam(fstRsdAmp)`	Table 9-18
`nSubbandScf[subframe] = nPrevious`	
`while (subframe != 7){`	
`// get subframe distance to next interpolation point`	
`nextSubframeDistance = getVariableParam(rsdAppx) + 1`	Table 9-18
`// get value of next interpolation point`	
`nNext = getVariableParam(rsdAmp)`	Table 9-18
`if (nNext & 1)`	
`nNext = nPrevious + ((nNext + 1) >> 1)`	
`else`	
`nNext = nPrevious - (nNext >> 1)`	
`// perform linear interpolation on missing values`	

syntax	Size (Bits)

`if (nextSubframeDistance = 2){`
`  if (nNext > nPrevious)`
`  nSubbandScf[subframe + 1] = nPrevious + (nNext - nPrevious) >> 1`
`  else`
`  nSubbandScf[subframe + 1] = nPrevious - (nPrevious - nNext) >> 1`
`}`
`else if (nextSubframeDistance = 4){`
`  if (nNext > nPrevious){`
`  nSubbandScf[subframe + 1] = nPrevious + (nNext - nPrevious) >> 2`
`  nSubbandScf[subframe + 2] = nPrevious + (nNext - nPrevious) >> 1`
`  nSubbandScf[subframe + 3] = nPrevious + ((nNext - nPrevious) * 3) >> 2`
`  }`
`  else{`
`  nSubbandScf[subframe + 1] = nPrevious - (nPrevious - nNext) >> 2`
`  nSubbandScf[subframe + 2] = nPrevious - (nPrevious - nNext) >> 1`
`  nSubbandScf[subframe + 3] = nPrevious - ((nPrevious - nNext) * 3) >> 2`
`} }`
`else{`
`  for (i = 1; i < nextSubframeDistance; i++){`
`  nSubbandScf[subframe + i] = nPrevious + (nNext - nPrevious)`
`  * i / nextSubframeDistance`
`} }`
`// final interpolation point`
`nSubbandScf[subframe + nextSubframeDistance] = nNext`
`// move on to next subframe`
`subframe += nextSubframeDistance`
`// update`
`nPrevious = nNext`
`} }`	

where:

fstRsdAmp, rsdAppx and rsdAmp are defined in clause 9.9.10.

Table 9-30: Decode Grid3

syntax	Size (Bits)
`DecodeGrid3(){`	
`  for (i = 0; i < 8; i++){`	
`  nValue = ExtractBits(5)`	5
`  r = Grid3Codes[nValue]`	
`  r &= 31`	
`  if(r<31)`	
`  return r`	
`  if( ExtractBits(1) = 0 )`	1
`  return 0xd`	
`  if( ExtractBits(3) = 3 )`	3
`  return 0x13`	
`  if( ExtractBits(7) = 1 )`	7
`  return 0xc`	
`  if( ExtractBits(1) = 1 )`	1
`  return 0x14`	
`  if( ExtractBits(3) = 0 )`	3
`  return 0xb`	
`  if( ExtractBits(7) = 6 )`	7
`  return 0x15`	
`  if( ExtractBits(15) = 2 )`	15
`  return 0xa`	
`  // rare value`	
`  nBits = ExtractBits(3) + 4`	3
`  r = ExtractBits(nBits)`	nbits
`  if(r > 56)`	
`  return 16`	
`  return r`	
`} }`	

where:

```
Grid3Codes[32] =
{
   113,  48,   79,   48,   142,  48,   79,   48,113,  48,   79,   48,   178,  48,   79,   48,
   113,  48,   79,   48,   142,  48,   79,   48,113,  48,   79,   48,   191,  48,   79,   48
}
```

Table 9-31: DecodeLPC

syntax	Size (Bits)
DecodeLPC(startSB,endSB){	
nOrder = 8	
for (nSubband = startSB; nSubband < endSB; nSubband++){	
for (nCh = ch; nCh <= (ch+1); nCh++){	
if (nSubband < 2){ // LPC subbands	
for (j = 0; j < 2; j++){ // LPC frames	
if (nBitsLeft >= nOrder * 4){	
for (i = 0; i < nOrder; i++){	
LpcQtd[i] = ExtractBits(4)	4
} } } }	
else{	
if (nBitsLeft >= nOrder * 4)	
{	
for (i = 0; i < nOrder; i++){	
LpcQtd[i] = ExtractBits(4)	4
} } } } } }	

Table 9-32: Decode Residual Chunks Part 2

syntax	Size (Bits)
ResidualChunksPart2(ch, nSubFrameCount)	
{	
nSampleOffset = nSubFrameCount << 3	
nTotalSubbands = 8 << nFrequencyRange	
nSubFrame4 = nSubFrameCount/4	
for (nSubband = 0; nSubband < nTotalSubbands ; nSubband++){	
if ((nSubband >= nMinMonoSubband) & (SecChPres = 0))	
nCodedChannels = ch	
else	
nCodedChannels = ch+1	
for (nCh = ch; nCh <= nCodedChannels; nCh++){	
nLevel = QLevels[nCh][nSubband][nSubFrame4]	
ModuloOrigin = 0	
if(nLevel = 2){	
if (nBitsLeft >= 16){	
if (TSCodingMethod[nCh][nSubband][nSubFrame4] = 0){	
nValue = nPackedSamples[nCh][nSubband]	
for (j = 0; j < 8; j++){	
nCount = mod(5, nSampleOffset - ModuloOrigin + j)	
if (nCount=0){	
if (nBitsLeft >= 24)	
nValue = ExtractBits(8)	8
else if (nBitsLeft >= 8){	
nValue = ExtractBits(8)	8
if (nValue > 242) nValue = 121	
}else	
nValue = 121	
}	
xValue[j] =	
ResidualLevels3[(ResidualPack5In8[nValue]>>(nCount*2))&3]	
}	
nPackedSamples[nCh][nSubband] = nValue	
}else{	
for (j = 0; j < 8; j++){	
nValue = ExtractBits(1)	1
if (nValue = 0)	

syntax	Size (Bits)
xValue[j] = 0	
else	
xValue[j] = ResidualQuantizednLevel16[nValue]	
} } }	
}else if (nLevel = 3){	
if (nBitsLeft >= 7){	
nValue = nPackedSamples[nCh][nSubband]	
if (nValue > 124) nValue = 124	
for (j = 0; j < 8; j++){	
nCount = mod(3,[nSampleOffset - ModuloOrigin + j])	
if (nCount=0){	
if (nBitsLeft >= 24)	
nValue = ExtractBits(7)	7
}else if (nBitsLeft >= 7){	
nValue = ExtractBits(7)	7
if (nValue > 124) nValue = 62	
}else{	
nValue = 62	
} }	
xValue[j] = ResidualLevels5[ResidualPack3In7[nValue][nCount]]	
}	
nPackedSamples[nCh][nSubband] = nValue	
} }	
else {	
switch (nLevel) {	
default:	
case 0:	
break	
case 1:	
if (nBitsLeft >= 8){	
nValue = ExtractBits(8)	8
xValue[0] = ResidualQuantizednLevel10[nValue & 1]	
nValue >>= 1	
xValue[1] = ResidualQuantizednLevel10[nValue & 1]	
nValue >>= 1	
xValue[2] = ResidualQuantizednLevel10[nValue & 1]	
nValue >>= 1	
xValue[3] = ResidualQuantizednLevel10[nValue & 1]	
nValue >>= 1	
xValue[4] = ResidualQuantizednLevel10[nValue & 1]	
nValue >>= 1	
xValue[5] = ResidualQuantizednLevel10[nValue & 1]	
nValue >>= 1	
xValue[6] = ResidualQuantizednLevel10[nValue & 1]	
nValue >>= 1	
xValue[7] = ResidualQuantizednLevel10[nValue & 1]	
}	
break	
case 4:	
for (j=0; j < 8; j++){	
if (nBitsLeft >= 24)	
xValue[j] = ResidualLevels8[nCodeVal[nValue & 63]]	
}	
break	
case 5:	
for (j=0; j < 8; j++){	
if (nBitsLeft >= 4)	
xValue[j] = ResidualLevels16[ExtractBits(4)]	4
}	
break	
} } } } }	

where:

The ResidualLevels and ResidualQuantizednLevel tables are in clause 9.9.5.

mod(x,y) is the modulo of y to the rank of x, so mod(x,y) is the integer remainder of y/x

```
ResidualPack5In8[256] =
{
  0x0000, 0x0100, 0x0200, 0x0040, 0x0140, 0x0240, 0x0080, 0x0180, 0x0280, 0x0010, 0x0110, 0x0210,
  0x0050, 0x0150, 0x0250, 0x0090, 0x0190, 0x0290, 0x0020, 0x0120, 0x0220, 0x0060, 0x0160, 0x0260,
  0x00a0, 0x01a0, 0x02a0, 0x0004, 0x0104, 0x0204, 0x0044, 0x0144, 0x0244, 0x0084, 0x0184, 0x0284,
  0x0014, 0x0114, 0x0214, 0x0054, 0x0154, 0x0254, 0x0094, 0x0194, 0x0294, 0x0024, 0x0124, 0x0224,
  0x0064, 0x0164, 0x0264, 0x00a4, 0x01a4, 0x02a4, 0x0008, 0x0108, 0x0208, 0x0048, 0x0148, 0x0248,
  0x0088, 0x0188, 0x0288, 0x0018, 0x0118, 0x0218, 0x0058, 0x0158, 0x0258, 0x0098, 0x0198, 0x0298,
  0x0028, 0x0128, 0x0228, 0x0068, 0x0168, 0x0268, 0x00a8, 0x01a8, 0x02a8, 0x0001, 0x0101, 0x0201,
  0x0041, 0x0141, 0x0241, 0x0081, 0x0181, 0x0281, 0x0011, 0x0111, 0x0211, 0x0051, 0x0151, 0x0251,
  0x0091, 0x0191, 0x0291, 0x0021, 0x0121, 0x0221, 0x0061, 0x0161, 0x0261, 0x00a1, 0x01a1, 0x02a1,
  0x0005, 0x0105, 0x0205, 0x0045, 0x0145, 0x0245, 0x0085, 0x0185, 0x0285, 0x0015, 0x0115, 0x0215,
  0x0055, 0x0155, 0x0255, 0x0095, 0x0195, 0x0295, 0x0025, 0x0125, 0x0225, 0x0065, 0x0165, 0x0265,
  0x00a5, 0x01a5, 0x02a5, 0x0009, 0x0109, 0x0209, 0x0049, 0x0149, 0x0249, 0x0089, 0x0189, 0x0289,
  0x0019, 0x0119, 0x0219, 0x0059, 0x0159, 0x0259, 0x0099, 0x0199, 0x0299, 0x0029, 0x0129, 0x0229,
  0x0069, 0x0169, 0x0269, 0x00a9, 0x01a9, 0x02a9, 0x0002, 0x0102, 0x0202, 0x0042, 0x0142, 0x0242,
  0x0082, 0x0182, 0x0282, 0x0012, 0x0112, 0x0212, 0x0052, 0x0152, 0x0252, 0x0092, 0x0192, 0x0292,
  0x0022, 0x0122, 0x0222, 0x0062, 0x0162, 0x0262, 0x00a2, 0x01a2, 0x02a2, 0x0006, 0x0106, 0x0206,
  0x0046, 0x0146, 0x0246, 0x0086, 0x0186, 0x0286, 0x0016, 0x0116, 0x0216, 0x0056, 0x0156, 0x0256,
  0x0096, 0x0196, 0x0296, 0x0026, 0x0126, 0x0226, 0x0066, 0x0166, 0x0266, 0x00a6, 0x01a6, 0x02a6,
  0x000a, 0x010a, 0x020a, 0x004a, 0x014a, 0x024a, 0x008a, 0x018a, 0x028a, 0x001a, 0x011a, 0x021a,
  0x005a, 0x015a, 0x025a, 0x009a, 0x019a, 0x029a, 0x002a, 0x012a, 0x022a, 0x006a, 0x016a, 0x026a,
  0x00aa, 0x01aa, 0x02aa, 0x0003, 0x0103, 0x0203, 0x0043, 0x0143, 0x0243, 0x0083, 0x0183, 0x0283,
  0x0013, 0x0113, 0x0213, 0x0053
}

ResidualPack3In7[128][3] =
{
 { 0, 0, 0 }, { 0, 0, 1 }, { 0, 0, 2 }, { 0, 0, 3 }, { 0, 0, 4 }, { 0, 1, 0 }, { 0, 1, 1 },
 { 0, 1, 2 }, { 0, 1, 3 }, { 0, 1, 4 }, { 0, 2, 0 }, { 0, 2, 1 }, { 0, 2, 2 }, { 0, 2, 3 },
 { 0, 2, 4 }, { 0, 3, 0 }, { 0, 3, 1 }, { 0, 3, 2 }, { 0, 3, 3 }, { 0, 3, 4 }, { 0, 4, 0 },
 { 0, 4, 1 }, { 0, 4, 2 }, { 0, 4, 3 }, { 0, 4, 4 }, { 1, 0, 0 }, { 1, 0, 1 }, { 1, 0, 2 },
 { 1, 0, 3 }, { 1, 0, 4 }, { 1, 1, 0 }, { 1, 1, 1 }, { 1, 1, 2 }, { 1, 1, 3 }, { 1, 1, 4 },
 { 1, 2, 0 }, { 1, 2, 1 }, { 1, 2, 2 }, { 1, 2, 3 }, { 1, 2, 4 }, { 1, 3, 0 }, { 1, 3, 1 },
 { 1, 3, 2 }, { 1, 3, 3 }, { 1, 3, 4 }, { 1, 4, 0 }, { 1, 4, 1 }, { 1, 4, 2 }, { 1, 4, 3 },
 { 1, 4, 4 }, { 2, 0, 0 }, { 2, 0, 1 }, { 2, 0, 2 }, { 2, 0, 3 }, { 2, 0, 4 }, { 2, 1, 0 },
 { 2, 1, 1 }, { 2, 1, 2 }, { 2, 1, 3 }, { 2, 1, 4 }, { 2, 2, 0 }, { 2, 2, 1 }, { 2, 2, 2 },
 { 2, 2, 3 }, { 2, 2, 4 }, { 2, 3, 0 }, { 2, 3, 1 }, { 2, 3, 2 }, { 2, 3, 3 }, { 2, 3, 4 },
 { 2, 4, 0 }, { 2, 4, 1 }, { 2, 4, 2 }, { 2, 4, 3 }, { 2, 4, 4 }, { 3, 0, 0 }, { 3, 0, 1 },
 { 3, 0, 2 }, { 3, 0, 3 }, { 3, 0, 4 }, { 3, 1, 0 }, { 3, 1, 1 }, { 3, 1, 2 }, { 3, 1, 3 },
 { 3, 1, 4 }, { 3, 2, 0 }, { 3, 2, 1 }, { 3, 2, 2 }, { 3, 2, 3 }, { 3, 2, 4 }, { 3, 3, 0 },
 { 3, 3, 1 }, { 3, 3, 2 }, { 3, 3, 3 }, { 3, 3, 4 }, { 3, 4, 0 }, { 3, 4, 1 }, { 3, 4, 2 },
 { 3, 4, 3 }, { 3, 4, 4 }, { 4, 0, 0 }, { 4, 0, 1 }, { 4, 0, 2 }, { 4, 0, 3 }, { 4, 0, 4 },
 { 4, 1, 0 }, { 4, 1, 1 }, { 4, 1, 2 }, { 4, 1, 3 }, { 4, 1, 4 }, { 4, 2, 0 }, { 4, 2, 1 },
 { 4, 2, 2 }, { 4, 2, 3 }, { 4, 2, 4 }, { 4, 3, 0 }, { 4, 3, 1 }, { 4, 3, 2 }, { 4, 3, 3 },
 { 4, 3, 4 }, { 4, 4, 0 }, { 4, 4, 1 }, { 4, 4, 2 }, { 4, 4, 3 }, { 4, 4, 4 }, { 5, 0, 0 },
 { 5, 0, 1 }, { 5, 0, 2 }
}

nCodeVal[64]=
{
 1, 2, 4, 3, 5, 2, 0, 3, 1, 2, 4, 3, 6, 2, 0, 3, 1, 2, 4, 3, 5, 2, 0, 3, 1, 2, 4, 3, 7, 2, 0, 3,
 1, 2, 4, 3, 5, 2, 0, 3, 1, 2, 4, 3, 6, 2, 0, 3, 1, 2, 4, 3, 5, 2, 0, 3, 1, 2, 4, 3, 4, 2, 0, 3
}
```

9.6.3.3.2 Quantization Profiles

The amount of bits available to store (quantized) information on individual values of subband samples varies from frame to frame. The actual amount depends on many variables: bitrate, number of bits occupied by tonal information, number of bits occupied by residual scalefactor grids. In these conditions it is feasible to use individual quantization profile in each frame. For multi-channel files, a quantization profile can be selected individually for each channel pair. The quantization profile is characterized by 8 bit value with the following fields:

- 3 bits - overall level;
- 2 bits - steepness;
- 3 bits - subband with maximum energy out of the first 8 subbands.

Quantization level for a given subband can be calculated with the following formulas:

$$SNR = \left\lfloor \frac{18000}{\left\lfloor \frac{12 \times nSubband \times SampleRate}{1000 \times TotalSubbands} \right\rfloor + 100 + 40 \times S} \right\rfloor + 20 \times L$$

where:

SNR - desired signal-to-noise ratio for given subband, in dB × 10

SampleRate - sampling rate in Hz;

ST - steepness;

OL - overall level.

A quantizer is then selected that give SNR value closest to calculated by the equation above according to:

Table 9-33: Quantizer levels

SNR, dB × 10	Number of levels in quantizer
≤ 95	2
> 95 and ≤ 140	3
> 140 and ≤ 180	5
> 180 and ≤ 230	8
> 230	16

9.6.3.3.3 Scale Factor Processing

At the encoder, the initial scale factor grid is referred to as the high resolution grid and is constructed by selecting the larger amplitude out of each pair of successive samples. This results in 32 bands by 64 scale factors per primary channel, but this grid is not transmitted due to its large size. Instead, 3 separate lower resolution grids are derived from it and encoded for packing into the bit stream.

At the decoder, the low resolution grids are decoded, re-assembled and then used to reconstruct a high resolution scale factor grid for sample scaling. If the 'high resolution grids' chunk is present, it is possible to construct scalefactors with better resolution. This chunk contains the 1st subband from Grid1 which should be processed first. Both Grid2 and Grid3 information is used to correct the results calculated using only Grid1 information. That is, Grid2 and Grid3 contain difference information for the Grid1 information.

Below is a summary of the frequency and time resolution of each grid at 44,1/48 KHz, (refer to the appropriate tables to derive 96 KHz values.)

Grid 1 has 10 bands of 8 scale factors, (see clause 9.9.4, Grid1 mapping table, for details). It should be noted that a skip and interpolate technique is also used at the encoder to improve coding efficiency, so the actual number of Grid1 factors in the stream could be less than 10 by 8:

- Bands 0 to 3 correspond to frequency subband 0 to 3 respectively
- Band 4: frequency subbands 4 to 5
- Band 5: frequency subbands 6 to 9
- Band 6: frequency subbands 7 to 12
- Band 7: frequency subbands 10 to 17
- Band 8: frequency subbands 14 to 23
- Band 9: frequency subbands 19 to 29

Grid 2 has 3 bands of 64 scale factors:

- Band 1: frequency subbands 4 to 9
- Band 2: frequency subbands 10 to 17
- Band 3: frequency subbands 18 to 31

Grid 3 has 26 bands of 8 scale factors:

- Mapping to frequency subbands 4 to 29

9.6.3.3.4 Decoding of Grid 1 scale factors

Grid1 has overlapping subbands (as opposed to Grid2 and Grid3). The bitstream contains amplitudes of selected Grid1 scalefactors and the distances between them. Any factors missing from the bit stream should be linearly interpolated from these values to produce 8 scalefactors per subband.

These values should further be mapped to the high-resolution grid using the weights from the tables in clause 9.9.4. The Grid1 chunk in the bit stream has information about all the Grid1 subbands except the first and second subband because for coding at low bitrates where there are no high resolution grids and no TimeSamples chunks in the bitstream, this subband is always zero.

9.6.3.3.5 Decoding of Grid 2 scale factors

Grid2 scalefactors are stored in groups of 8. There is a 1bit flag stored before each group - if this flag is 0 - the whole group has 0 values. Otherwise individual values are stored. These values should be subtracted from the high-resolution grid. Grid2 has the same time resolution as the high-resolution grid, but there are only three subbands. These three subbands are mapped without weighting to high-resolution bands 4 to 7; 8 to 15; and 16 to 31 respectively.

9.6.3.3.6 Decoding of Grid 3 scale factors

Grid3 has the same frequency resolution as the high-resolution grid but has low time-resolution - there are only 8 time intervals. Since Grid1 already has the highest possible frequency resolution for the lowest 4 subbands, there is no data for these subbands in Grid3. Due to this, Grid3 contains only 28 subbands which should be mapped to subbands 4 to 31 of the high resolution grid.

9.6.3.4 Reconstruction of Hi resolution scale factors grid

High-resolution scalefactors are constructed from Grid1 and 'high resolution grids' chunks. In this step all the grids obtained from the bitstream are mapped onto a grid of scalefactors which has enough time and frequency resolution to include information from all the grids. A grid 64 time intervals is sufficient. In the clauses that follow, this grid will be referred as the 'high-resolution grid'.

9.6.3.5 LPC synthesis

The synthesis process contains the following steps (for each subband):

1) Getting reflection coefficients (LpcQtd[] in Table 9-31) from bitstream.
2) Dequantizing coefficients to direct linear representation.
3) Running predictor over the samples in a form of IIR filter.

For each primary channel, prediction is applied to the first two subbands only and each frame is divided into two blocks for processing. So there would be two groups of 8 coefficients per subband in the bit stream and each set should be used to synthesize samples for half of the frame.

9.6.3.6 Timesamples Processing

There are several methods for packing time sample information into the bitstream. These methods are indicated with flags that apply to a single subband for the entire residual frame:

- If nLRMS (Mid/Side)is set, the time samples are used for both channels.
- If nSubbandMidSide is set in conjunction with nLRMS , the time samples for the right channel should be inverted.
- The TSCodingMethod flag determines whether time samples are packed using Huffman codes or packed directly as 5 values in 8 bits for 3-level quantization or as 3 values in 7 bits for 5-level quantization. This flag is effective only for 3- and 5-level quantization. 9- and 8-level quantization always uses Huffman codes. 2-level quantization always uses direct packing (1bit/sample).

De-quantized values are described in the tables found in clause 9.9.5.

The quantization levels allocation scheme used to quantize the time-samples depends on quantization profile that is selected individually for each frame and each channel pair. A procedure of assignment of quantization levels to subbands is described in clause 9.6.3.3.2.

When no information about specific sample value is found in the bitstream, a pseudo-random value with the range [-1…1] is substituted. This process is referred to as white noise substitution.

After de-quantization, the time samples are scaled with the high-resolution scale factors (see clause 9.6.3.3.3). There are 2 samples for each scale factor in the high-resolution scale factors grid. Each time sample should be multiplied by the linear scale factor value obtained from the Grid 1, 2 and 3 scale factor extraction.

9.6.4 Inverse Filterbank

The inverse filterbank is constructed of evenly spaced critically downsampled subbands. The filterbank is a hybrid structure performed with the following steps:

1) Windowing input data in each subband:
   - 8-point forward MDCT
   - Grouping 8-point MDCT results into a single set of MDCT coefficients
   - Aliasing cancellation for high frequencies, described by the following pseudo-code:

```
AL1 = 0.30865828381746
AL2 = 0.03806023374436
a = mdct_band[i][3]   * AL1
b = mdct_band [i+1][0] * AL1
mdct_band [i][3]   = mdct_band [i][3]   + b-a
mdct_band [i+1][0] = mdct_band [i+1][0] + b+a
a = mdct_band [i][2]   * AL2
b = mdct_band [i+1][1] * AL2
mdct_band [i][2]   = mdct_band [i][2]   + b-a
mdct_band [i+1][1] = mdct_band [i+1][1] + b+a
```

Where `mdct_band[i]` is i^{th} band of MDCT coefficients, and the short window filter is shown in Table 9-34.

Table 9-34: Short window filter

0,02281089288256
0,41799773023326
0,90844807089885
0,99973979773034
0,99973979773034
0,90844807089885
0,41799773023326
0,02281089288256

N-point inverse MDCT. N depends on sample-rate as shown in Table 9-19.

2) Window (Long Window in clause 9.9.6).

3) Overlap-add.

This inverse filterbank structure allows an optimized implementation to use the same filterbank for both residual coding and tonal components reconstruction, which speeds-up the decoding process considerably.

9.6.5 LFE Chunk

9.6.5.1 LFE Chunk Syntax

The LFE Chunk is used to store ADPCM-encoded LFE channel samples.

Table 9-35: LFE Chunk

Syntax	Size (Bits)
`LFEChunk()`	
`{`	
`    if (chunkID != LFE_ADPCM_ID)`	8
`        return`	
`    ChunkLengthInfo()`	Table 9-9
`    DecodeLFE()`	
`    ByteAlign()`	0..7
`}`	

Table 9-36: Decode LFE

Syntax	Size (Bits)
`DecodeLFE(){`	
`    InitLFE()`	Table 9-37
`    vp = 0`	
`    nsamples = 4`	
`    stepindex = 0`	
`    if (nSampleRate<14000){`	
`        upsampleFactor = 16`	
`    }else   if (nSampleRate<28000){`	
`        upsampleFactor = 32`	
`    }else if (nSampleRate<50000){`	
`        upsampleFactor = 64`	
`    }else{`	
`        upsampleFactor = 128`	
`    }`	
`    nScale = upsampleFactor * 0x7fff`	
`    vp = predictedSampleInit`	
`    stepindex = stepSizeIndexInit`	
`    if (bLFEinput24Bit){`	
`        stepsize = lfe_StepSizeTable24[stepindex]`	
`        for (nsample = 0; nsample<nsamples; nsample++){`	
`            code = ExtractBits(6)`	6
`            // Calculate predicted delta as ((code+0.5)*step)/16`	
`            pdelta = stepsize/32`	
`            if (code & 16)`	
`                pdelta += stepsize`	
`            if (code & 8)`	
`                pdelta += stepsize/2`	
`            if (code & 4)`	
`                pdelta += stepsize/4`	
`            if (code & 2)`	
`                pdelta += stepsize/8`	
`            if (code & 1)`	
`                pdelta += stepsize/16`	
`            // Update predicted value`	
`            if (code & 32){`	
`                vp -= pdelta`	
`                if (vp < -1.3f)`	

Syntax	Size (Bits)

vp = -1.3f
}else{
vp += pdelta
if (vp > 1.3f)
vp = 1.3f
}
// Adjust step size
stepindex += lfe_DeltaIndex24[code & 31]
if (stepindex < 0){
stepindex = 0
}else if (stepindex > 143){
stepindex = 143
}
stepsize = lfe_StepSizeTable24[stepindex]
// Output data
LFEdata[nsample] = vp * nScale
} }
else
{
stepsize = lfe_StepSizeTable16[stepindex]
for (nsample = 0; nsample<nsamples; nsample++){
code = ExtractBits(4)
	4

// Calculate predicted delta as ((code+0.5)*step)/4
pdelta = stepsize/8
if (code & 4)
pdelta += stepsize
if (code & 2)
pdelta += stepsize/2
if (code & 1)
pdelta += stepsize/4
// Update predicted value
if (code & 8){
vp -= pdelta
if (vp < -1.3f)
vp = -1.3f
}else{
vp += pdelta
if (vp > 1.3f)
vp = 1.3f
}
// Adjust step size
stepindex += lfe_DeltaIndex16[code & 7]
if (stepindex < 0){
stepindex = 0
}else if (stepindex > 100){
stepindex = 100
}
stepsize = lfe_StepSizeTable16[stepindex]
// Output data
LFEdata[nsample] = vp * nScale
} }	

where:

lfe_DeltaIndex16[] and *lfe_DeltaIndex24*[] are in clause 9.9.7,

lfe_StepSizeTable16[] and *lfe_StepSizeTable24*[] are in clause 9.9.8.

Table 9-37: Init LFE Decoding

Syntax	Size (Bits)
InitLFE(){	
// determining input bit depth from chunk size	
bLFEinput24Bit = (bLBRCompressedFlags && LBR_FLAG_24_BIT_SAMPLES)	
if (bLFEinput24Bit){	
int_ps = ExtractBits(8)	8
int_ps |= ExtractBits(16) << 8	16
ps_mult = 0x007fffff	
if (int_ps & 0x00800000){	
int_ps &= 0x007fffff	
ps_mult = -ps_mult	
}	
}else{	
int_ps = ExtractBits(16)	16
ps_mult = 0x007fff	
if (int_ps & 0x008000){	
int_ps &= 0x007fff	
ps_mult = -ps_mult	
}	
}	
predictedSampleInit = int_ps/ps_mult	
stepSizeIndexInit = ExtractBits(8)	8
}	

The number of bits used for *DeltaValue* and *StartStepSizeIndex* depends on the source data resolution: 3 bits for 16-bit source, 5 bits for 24-bit source.

9.6.5.2 LFE decoding

LFE decoding is performed whenever LFE channel information is present in the bitstream. ADPCM-encoded LFE coefficients are extracted from the bitstream and ADPCM synthesis is performed. A further upsampling is required to match the LFE channel sample rate to the rest of the channels:

- 64 times for sample rates $\leq$ 48 kHz;
- 128 times for sample rates > 48 kHz and $\leq$ 96 kHz; and
- 256 times for sample rates > 96 kHz.

ADPCM synthesis process

ADPCM samples are stored to the bitstream as fixed-length numbers. 4 bits are used for 16-bit source samples and 6 bits are used for 24-bit samples. A simple ADPCM method, linearly predicting DPCM step size from the difference between previous two samples is implemented.

The current sample value V is calculated by one of the following formula:

$$V = Vp + (-1)Sign \times \frac{(Code + 0{,}5) \times StepSize}{4} \qquad \text{for 16-bit samples}$$

$$V = Vp + (-1)Sign \times \frac{(Code + 0{,}5) \times StepSize}{16} \qquad \text{for 24-bit samples}$$

where:

Code is the part of encoded sample which shows the absolute ratio of the difference between the current and the previous samples to *StepSize*.

Sign is the sign of the difference between the current and the previous samples.

Vp is the value of the previous decoded sample.

StepSize is the step size

StepSizeIndex is the index of the *StepSize* in the Step Size Table.

At the end of the step, new *Vp* and *StepSize* values are calculated:

$Vp' = V$.

$StepSizeIndex' = StepSizeIndex + DeltaIndexTable[Code]$.

$StepSize'$ = StepSizeTable[StepSizeIndex'].

See the appropriate tables in clause 9.9.7 for *DeltaIndex* table values and in clauses 9.9.8 for *StepSize* table values.

StepSizeIndex and *Vp* are read at the start of each frame to ensure frame-drop tolerance.

Initial StepSize is *StepSize* Table[0].

9.6.6 Embedded Channel Sets Chunk

9.6.6.1 About the Embedded Channel Sets Chunk

The Embedded Channel Sets (ECS) chunk holds residual samples inter-channel replacement information in the case of stereo downmix (LBR_FLAG_STEREO_DOWNMIX is set) and downmix scaling and contribution coefficients in the case of multi-channel downmix (LBR_FLAG_MULTICHANNEL_DOWNMIX is set).

9.6.6.2 Embedded channel sets

Whenever the embedded channel sets chunk is present in the bitstream, additional operations are needed to extract the original channels. The required information is obtained directly from the embedded channel sets chunk.

There are two possible scenarios:

- Decoders capable of decoding no more than 2 channels: will decode the first two channels, ignoring irrelevant information in the tonal chunk and irrelevant residual chunks.
- Decoders capable of decoding 5.1 channels: will decode 5 channels plus the LFE channel.

9.6.6.3 Stereo downmix case

Whenever a non-zero number is present for replacementChannel in ECSChunk(), a corresponding number of timesamples have to be taken from that replacementChannel (which is one of the downmixed channels). For each channel and for each subband there is a single entry in ECSChunk which covers a quarter of a frame e.g. four replacementChannel values per frame duration.

Table 9-38: Embedded Channel Set Chunk

Syntax	Size (Bits)
ECSChunk()	
{	
if (chunkID != EmbLevels_ID)	8
return	
ChunkLengthInfo()	Table 9-9
if (LBR_FLAG_USE_LFE)	
nFullbandChannels = nChannels + 1	
else	
nFullbandChannels = nChannels	
// there are two types of information in this chunk:	
// - replacement information for stereo downmix, or	
// - scaling/contribution info for multi-channel downmix	
if (replacementPair){	
stSB = ExtractBits(7)	7
enSB = ExtractBits(7)	7
for (c=0; c < nChannels * nTotalSubbands * 4; ++c){	

Syntax	Size (Bits)
nValue = ExtractBits(1)	1
if(nValue){	
canReplace = c	
nValue = ExtractBits(1)	1
if(nValue){	
replaceChannel = c	
} } } }	
else	
{	
if (LBR_FLAG_MULTICHANNEL_DOWNMIX)	
bMultichannelDownMix = TRUE	
else	
bMultichannelDownMix = FALSE	
if (LBR_FLAG_CS_IS_LAST_DOWNMIX)	
nCsIsLast = 1	
else	
nCsIsLast = 0	
nContChAenc = ExtractBits(4)	4
nContChBenc = ExtractBits(4)	4
if (nChannels >= 2+6)	
nContChA = ContChOrderForDecode[nFullbandChannels-2-6][nChannels - nFullbandChannels][nContChAenc]	
if (nContChBenc >= (nFullbandChannels + (bMultichannelDownMix)?0:2)){	
nContChB = -1	
nContChBenc = -1	
}else{	
// there is no need to compute indexes of contributed channels for decoder	
// if we're only decoding 5.x (or 4.x) embedded downmix	
if (nFullbandChannels >= 2+6)	
nContChB = ContChOrderForDecode[nFullbandChannels-2-6][nChannels - nFullbandChannels][nContChBenc]	
}	
// Read 4 contribution coeffs	
for (c = 0; c < 4-(2*nCsIsLast); c++){	
nDMixCoeffIndex = ExtractBits(6)	6
DMixContributedCoeffs[c] = DMixContribution_IndexTodB[nDMixCoeffIndex]	
}	
// Read 1 or all down-mix channels scaling coeffs	
nValue = ExtractBits(1)	1
if (nValue){	
for (c=0; c < nChannels - ((bMultichannelDownMix)?4:0); c++){	
nDMixCoeffIndex = ExtractBits(5)	5
DMixScalingCoeffs[c] = DMixContribution_IndexTodB[nDMixCoeffIndex]	
DMixScalingCoeffs_dB[c] = DMixScalingCoeffs[c]	
if((DMixScalingCoeffs[c] = -200.0f) | (DMixScalingCoeffs[c] = 0.0f)){	
DMixScalingCoeffs[c] = 1.0f	
DMixScalingCoeffs_K[c] = 1.0f	
}else{	
DMixScalingCoeffs[c] = pow(10.0f, -DMixScalingCoeffs[c] / 20.0f)	
DMixScalingCoeffs_K[c] = DMixScalingCoeffs[c]	
} } }	
else {	
nDMixCoeffIndex = ExtractBits(5)	5
DMixScalingCoeffs[0] = DMixScaling_IndexTodB[nDMixCoeffIndex]	
for (c=0; c < nChannels - ((bMultichannelDownMix)?4:0); c++)	
DMixScalingCoeffs_dB[c] = DMixScalingCoeffs[0]	
if((DMixScalingCoeffs[0] = -200.0f) | (DMixScalingCoeffs[0] = 0.0f)){	
DMixScalingCoeffs[0] = 1.0f	
DMixScalingCoeffs_K[0] = DMixScalingCoeffs[0]	
}	
else {	
DMixScalingCoeffs[0] = pow(10.0f, -DMixScalingCoeffs[0] / 20.0f)	
DMixScalingCoeffs_K[0] = DMixScalingCoeffs[0]	
}	
for (c=1; c < nChannels - ((bMultichannelDownMix)?4:0); c++){	

Syntax	Size (Bits)

DMixScalingCoeffs[c] = DMixScalingCoeffs[0]
DMixScalingCoeffs_K[c] = DMixScalingCoeffs[0]
}
}
// LFE channel
if (nFullbandChannels != nChannels)
{
lfecoeff = DMixScalingCoeffs[nChannels - ((bMultichannelDownMix)?4:0)-1]
lfecoeff_K = DMixScalingCoeffs_K[nChannels - ((bMultichannelDownMix)?4:0)-1]
lfecoeff_dB = DMixScalingCoeffs_dB[nChannels -((bMultichannelDownMix)?4:0)-1]
for (c = nChannels - ((bMultichannelDownMix)?4:0)-1; c > 3; --c){
DMixScalingCoeffs[c] = DMixScalingCoeffs[c-1]
DMixScalingCoeffs_K[c] = DMixScalingCoeffs_K[c-1]
DMixScalingCoeffs_dB[c] = DMixScalingCoeffs_dB[c-1]
}
DMixScalingCoeffs[3] = lfecoeff
DMixScalingCoeffs_K[3] = lfecoeff_K
DMixScalingCoeffs_dB[3] = lfecoeff_dB
} }
ByteAlign()
}
	0..7

where:

DMixScaling_IndexTodB[] and *DMixContribution_IndexTodB[]* are presented in clause 9.9.9:

```
ContChOrderForDecode[2][2][6] =
{
   // Number of Dmix channels == 6
   {
      // no LFE
      {0, 1, 2, 3, 4, 5},
      // LFE exists
      {0, 1, 2, 4, 3, 5}
   },
   // Number of Dmix channels == 7
   {
   // no LFE
      {0, 1, 4, 2, 3, 5},
      // LFE exists
      {0, 1, 4, 5, 2, 3}
   }
}
```

9.7 Program Associated Data Chunk

The Program Associated Data Chunk can be used to store any data that is associated with the audio frame but is not required for decoding of the audio frame.

Table 9-39: Pad Chunk

Syntax	Size (Bits)

padChunk()
{
chunkID == padID
ChunkLengthInfo()
ProgramAssociatedData
ByteAlign()
}
	
	8
	Table 9-9
	variable
	0..7

9.8 Null Chunk

The Null Chunk is used to pad out the parent chunk. It is typically used when the parent chunk needs to be a constant size and the child chunks vary in size. The Null Chunk is different from other chunks in that the **chunkLength** field is optional if the Null Chunk is the last chunk within the parent chunk. It is recommended that the data field be filled with bytes of zero value.

Table 9-40: Null Chunk

Syntax	Size (Bits)

nullChunk()
{
NullID
[chunkLength]
NullData
}
	
	8
	16

9.9 Tables

9.9.1 Quantized Amplitude to Linear Amplitude Conversion

Table 9-41: Quantized Amplitude to Linear Amplitude Conversion

Quantized	Linear
0	0,17678
1	0,42678
2	0,60355
3	0,85355
4	1,20711
5	1,68359
6	2,375
7	3,36719
8	4,75
9	6,73438
10	9,5
11	13,46875
12	19
13	26,9375
14	38
15	53,875

Quantized	Linear
16	76
17	107,75
18	152
19	215,5
20	304
21	431
22	608
23	862
24	1216
25	1724
26	2432
27	3448
28	4864
29	6896
30	9728
31	13792

Quantized	Linear
32	19456
33	27584
34	38912
35	55168
36	77824
37	110336
38	155648
39	220672
40	311296
41	441344
33	27584
42	622592
43	882688
44	1245184
45	1765376
46	2490368

9.9.2 Wave synthesis envelope table

Table 9-42: Wave synthesis envelope table

index	wavSynEnvelope
0	0,00240763666390
1	0,00960735979838
2	0,02152983213390
3	0,03806023374436
4	0,05903936782582
5	0,08426519384873
6	0,11349477331863
7	0,14644660940673
8	0,18280335791818
9	0,22221488349020
10	0,26430163158700
11	0,30865828381746
12	0,35485766137277
13	0,40245483899194
14	0,45099142983522
15	0,50000000000000
16	0,54900857016478
17	0,59754516100806
18	0,64514233862723
19	0,69134171618254
20	0,73569836841300
21	0,77778511650980
22	0,81719664208182
23	0,85355339059327
24	0,88650522668137
25	0,91573480615127
26	0,94096063217418
27	0,96193976625564
28	0,97847016786610
29	0,99039264020162
30	0,99759236333610
31	1,00000000000000

index	wavSynEnvelope
32	0,99759236333610
33	0,99039264020162
34	0,97847016786610
35	0,96193976625564
36	0,94096063217418
37	0,91573480615127
38	0,88650522668137
39	0,85355339059327
40	0,81719664208182
41	0,77778511650980
42	0,73569836841300
43	0,69134171618255
44	0,64514233862723
45	0,59754516100806
46	0,54900857016478
47	0,50000000000000
48	0,45099142983522
49	0,40245483899194
50	0,35485766137277
51	0,30865828381745
52	0,26430163158700
53	0,22221488349020
54	0,18280335791818
55	0,14644660940673
56	0,11349477331863
57	0,08426519384873
58	0,05903936782582
59	0,03806023374436
60	0,02152983213390
61	0,00960735979838
62	0,00240763666390
63	0

9.9.3 Base function synthesis correction coefficients

Table 9-43: Base function synthesis correction coefficients

	Spectral line offset										
F_dlt	**-5**	**-4**	**-3**	**-2**	**-1**	**0**	**1**	**2**	**3**	**4**	**5**
0	-0,01179	0,04281	0,46712	0,46345	-3,94525	3,94525	-0,46345	-0,46712	-0,04281	0,01179	-0,00299
1	-0,00929	0,04882	0,45252	0,37972	-3,85446	4,03189	-0,55069	-0,4804	-0,03599	0,01445	-0,00229
2	-0,00696	0,05403	0,43674	0,29961	-3,75975	4,11413	-0,64135	-0,49221	-0,02834	0,01726	-0,00156
3	-0,00481	0,05847	0,41993	0,22319	-3,66138	4,19175	-0,73529	-0,50241	-0,01983	0,02021	-0,0008
4	-0,00284	0,06216	0,40224	0,15053	-3,55963	4,26452	-0,83239	-0,51085	-0,01047	0,02328	-0,00003
5	-0,00105	0,06515	0,38378	0,08168	-3,45475	4,33225	-0,93249	-0,51738	-0,00024	0,02646	0,00074
6	0,00054	0,06745	0,36471	0,01668	-3,34703	4,39475	-1,03543	-0,52184	0,01085	0,02973	0,00152
7	0,00195	0,06912	0,34515	-0,04445	-3,23676	4,45185	-1,14105	-0,5241	0,0228	0,03306	0,00228
8	0,00318	0,07017	0,32521	-0,10168	-3,12422	4,50339	-1,24914	-0,524	0,03561	0,03643	0,00302
9	0,00422	0,07065	0,30503	-0,15503	-3,00969	4,54921	-1,35952	-0,52141	0,04925	0,03981	0,00373
10	0,00508	0,07061	0,28471	-0,2045	-2,89348	4,58919	-1,47197	-0,51618	0,0637	0,04319	0,0044
11	0,00577	0,07007	0,26436	-0,25013	-2,77587	4,62321	-1,58627	-0,50818	0,07895	0,04652	0,00501
12	0,00629	0,06909	0,2441	-0,29194	-2,65716	4,65118	-1,70219	-0,49727	0,09494	0,04979	0,00556
13	0,00666	0,06769	0,224	-0,33	-2,53764	4,67302	-1,81949	-0,48335	0,11166	0,05295	0,00604
14	0,00687	0,06592	0,20416	-0,36435	-2,4176	4,68866	-1,93791	-0,46627	0,12904	0,05597	0,00642
15	0,00694	0,06383	0,18468	-0,39506	-2,29732	4,69806	-2,0572	-0,44593	0,14705	0,05881	0,00671
16	0,00689	0,06144	0,16561	-0,42223	-2,1771	4,7012	-2,1771	-0,42223	0,16561	0,06144	0,00689
17	0,00671	0,05881	0,14705	-0,44593	-2,0572	4,69806	-2,29732	-0,39506	0,18468	0,06383	0,00694
18	0,00642	0,05597	0,12904	-0,46627	-1,93791	4,68865	-2,41759	-0,36435	0,20416	0,06592	0,00687
19	0,00604	0,05295	0,11166	-0,48334	-1,81949	4,67301	-2,53763	-0,33	0,224	0,06769	0,00666
20	0,00556	0,04979	0,09494	-0,49727	-1,70219	4,65117	-2,65715	-0,29194	0,24409	0,06909	0,00629
21	0,00501	0,04652	0,07894	-0,50818	-1,58627	4,62321	-2,77587	-0,25013	0,26436	0,07007	0,00577
22	0,0044	0,04319	0,0637	-0,51618	-1,47197	4,58919	-2,89348	-0,2045	0,28471	0,07061	0,00508
23	0,00373	0,03981	0,04925	-0,52141	-1,35952	4,54921	-3,0097	-0,15503	0,30503	0,07065	0,00422
24	0,00302	0,03643	0,03561	-0,524	-1,24915	4,50339	-3,12422	-0,10168	0,32521	0,07017	0,00318
25	0,00228	0,03306	0,0228	-0,5241	-1,14105	4,45186	-3,23677	-0,04445	0,34515	0,06912	0,00195
26	0,00152	0,02973	0,01085	-0,52184	-1,03544	4,39477	-3,34704	0,01668	0,36471	0,06745	0,00054
27	0,00074	0,02646	-0,00024	-0,51738	-0,93249	4,33226	-3,45476	0,08168	0,38378	0,06515	-0,00105
28	-0,00003	0,02328	-0,01047	-0,51085	-0,83239	4,26452	-3,55963	0,15053	0,40224	0,06216	-0,00284
29	-0,0008	0,02021	-0,01983	-0,50241	-0,73529	4,19174	-3,66138	0,22319	0,41993	0,05847	-0,00481
30	-0,00156	0,01726	-0,02834	-0,49221	-0,64135	4,11413	-3,75974	0,29961	0,43674	0,05403	-0,00696
31	-0,00229	0,01445	-0,03599	-0,4804	-0,55069	4,03188	-3,85445	0,37972	0,45251	0,04882	-0,00929

9.9.4 Grid1 mapping tables

Table 9-44: Grid1 mapping table

High-resolution Grid subband	Grid1 subband number									
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	1	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0
2	0	0	1	0	0	0	0	0	0	0
3	0	0	0	1	0	0	0	0	0	0
4	0	0	0	0	0,5	0	0	0	0	0
5	0	0	0	0	0,5	0	0	0	0	0
6	0	0	0	0	0	0,5	0	0	0	0
7	0	0	0	0	0	0,33333	0,09524	0	0	0
8	0	0	0	0	0	0,16667	0,19048	0	0	0
9	0	0	0	0	0	0	0,28571	0	0	0
10	0	0	0	0	0	0	0,21429	0,05556	0	0
11	0	0	0	0	0	0	0,14286	0,11111	0	0
12	0	0	0	0	0	0	0,07143	0,16667	0	0
13	0	0	0	0	0	0	0	0,22222	0	0
14	0	0	0	0	0	0	0	0,17778	0,03636	0
15	0	0	0	0	0	0	0	0,13333	0,07273	0
16	0	0	0	0	0	0	0	0,08889	0,10909	0
17	0	0	0	0	0	0	0	0,04444	0,14545	0
18	0	0	0	0	0	0	0	0	0,18182	0
19	0	0	0	0	0	0	0	0	0,15152	0,02778
20	0	0	0	0	0	0	0	0	0,12121	0,05556
21	0	0	0	0	0	0	0	0	0,09091	0,08333
22	0	0	0	0	0	0	0	0	0,06061	0,11111
23	0	0	0	0	0	0	0	0	0,03030	0,13889
24	0	0	0	0	0	0	0	0	0	0,16667
25	0	0	0	0	0	0	0	0	0	0,13889
26	0	0	0	0	0	0	0	0	0	0,11111
27	0	0	0	0	0	0	0	0	0	0,08333
28	0	0	0	0	0	0	0	0	0	0,05556
29	0	0	0	0	0	0	0	0	0	0,02778

9.9.5 Quantization Levels for Residuals

Table 9-45: ResidualLevels16

Residual Code	Dequantized Value
0	-1,3125
1	-1,1375
2	-0,9625
3	-0,7875
4	-0,6125
5	-0,4375
6	-0,2625
7	-0,0875
8	0,0875
9	0,2625
10	0,4375
11	0,6125
12	0,7875
13	0,9625
14	1,1375
15	1,3125

Table 9-46: ResidualLevels8

Residual Code	Dequantized Value
0	-1,0
1	-0,625
2	-0,291666667
3	0,0
4	0,25
5	0,5
6	0,75
7	1,0

Table 9-47: ResidualLevels3

Residual Code	Dequantized Value
0	-0,645
1	0
2	0,645

Table 9-48: ResidualQuantizednLevel10

Residual Code	Dequantized Value
0	-0,47
1	0,47

Table 9-49: ResidualLevels5

Residual Code	Dequantized Value
0	-0,875
1	-0,375
2	0
3	0,375
4	0,875

Table 9-50: ResidualQuantizednLevel16

Residual Code	Dequantized Value
0	-0,645
1	0,645

9.9.6 Long window for filterbank

Only first 128 coefficients are tabulated since the rest of the table is symmetric.

Table 9-51: Long window for filterbank

1	0,00000073053931
2	0,00001971603402
3	0,00009120922576
4	0,00024999557063
5	0,00053053202896
6	0,00096681736288
7	0,00159226421635
8	0,00243957342599
9	0,00354061104012
10	0,00492628851495
11	0,00662644654387
12	0,00866974296265
13	0,01108354515765
14	0,01389382738765
15	0,01712507341094
16	0,02080018478972
17	0,02494039522219
18	0,02956519123018
19	0,03469223950626
20	0,04033732119902
21	0,04651427338908
22	0,05323493798151
23	0,06050911821199
24	0,06834454293557
25	0,07674683883740
26	0,08571951067489
27	0,09526392963020
28	0,10537932982169
29	0,11606281299140
30	0,12730936135520
31	0,13911185857091
32	0,15146111874875
33	0,16434592339804
34	0,17775306617356
35	0,19166740525504
36	0,20607192316413
37	0,22094779379428
38	0,23627445640116
39	0,25202969627406
40	0,26818973178253
41	0,28472930746762
42	0,30162179282255
43	0,31883928638529
44	0,33635272474356
45	0,35413199603255
46	0,37214605748681
47	0,39036305659014
48	0,40875045535117
49	0,42727515721822
50	0,44590363613336
51	0,46460206721486
52	0,48333645854690
53	0,50207278354759
54	0,52077711337935
55	0,53941574886118
56	0,55795535133868
57	0,57636307196621
58	0,59460667885545
59	0,61265468154600
60	0,63047645225724
61	0,64804234338508
62	0,66532380071377
63	0,68229347182117
64	0,69892530916529
65	0,71519466735162
66	0,73107839409342
67	0,74655491439262
68	0,76160430748517
69	0,77620837611373
70	0,79035070771131
71	0,80401672710242
72	0,81719374035394
73	0,82987096943593
74	0,84203957738371
75	0,85369268368674
76	0,86482536966764
77	0,87543467365609
78	0,88551957580828
79	0,89508097247291
80	0,90412164005964
81	0,91264618842693
82	0,92066100387146
83	0,92817418187376
84	0,93519544983175
85	0,94173608009759
86	0,94780879372230
87	0,95342765540680
88	0,95860796025748
89	0,96336611304655
90	0,96771950078316
91	0,97168635950719
92	0,97528563632273
93	0,97853684779033
94	0,98145993589285
95	0,98407512287726
96	0,98640276634975
97	0,98846321606187
98	0,99027667386693
99	0,99186305834592
100	0,99324187559754
101	0,99443209765490
102	0,99545204993079
103	0,99631930900172
104	0,99705061191921
105	0,99766177808460
106	0,99816764454334
107	0,99858201534882
108	0,99891762541818
109	0,99918611905815
110	0,99939804308295
111	0,99956285418553
112	0,99968893996391
113	0,99978365275329
114	0,99985335517798
115	0,99990347612183
116	0,99993857562689
117	0,99996241707224
118	0,99997804486209
119	0,99998786576711
120	0,99999373201709
121	0,99999702423632
122	0,99999873234653
123	0,99999953263198
124	0,99999985926787
125	0,99999996875111
126	0,99999999584044
127	0,99999999980564
128	0,99999999999973

9.9.7 Delta Index for LFE ADPCM

Table 9-52: Delta Index for 16-bit samples

Code	Index Adjustment
0	-4
1	-3
2	-2
3	-1
4	2
5	4
6	6
7	8

Table 9-53: Delta Index for 24-bit samples

Code	Index Adjustment
0	-8
1	-8
2	-7
3	-7
4	-6
5	-6
6	-5
7	-5
8	-4
9	-4
10	-3
11	-3
12	-2
13	-2
14	-1
15	-1
16	1
17	1
18	2
19	2
20	3
21	3
22	4
23	4
24	5
25	5
26	6
27	6
28	7
29	7
30	8
31	8

9.9.8 Step Size for LFE ADPCM encoding

Table 9-54: StepSize table for 16-bit samples

0	2,1362956633198035e-004
1	2,4414807580797754e-004
2	2,7466658528397473e-004
3	2,7466658528397473e-004
4	3,0518509475997192e-004
5	3,3570360423596911e-004
6	3,9674062318796350e-004
7	4,2725913266396069e-004
8	4,5777764213995788e-004
9	5,1881466109195227e-004
10	5,7985168004394665e-004
11	6,1037018951994385e-004
12	6,7140720847193823e-004
13	7,6296273689992981e-004
14	8,2399975585192419e-004
15	9,1555528427991577e-004
16	1,0071108127079073e-003
17	1,0986663411358989e-003
18	1,2207403790398877e-003
19	1,3428144169438765e-003
20	1,4648884548478652e-003
21	1,6174810022278512e-003
22	1,7700735496078372e-003
23	1,9531846064638203e-003
24	2,1362956633198035e-003
25	2,3499252296517838e-003
26	2,5940733054597613e-003
27	2,8687398907437361e-003
28	3,1434064760277108e-003
29	3,4485915707876827e-003
30	3,7842951750236518e-003
31	4,1810357982116153e-003
32	4,6082949308755760e-003
33	5,0660725730155339e-003
34	5,5543687246314890e-003
35	6,1037018951994385e-003
36	6,7445905941953795e-003
37	7,4159978026673177e-003
38	8,1484420300912512e-003
39	8,9419232764671782e-003
40	9,8574785607470940e-003
41	1,0834070863979004e-002
42	1,1932737205114903e-002
43	1,3122959074678793e-002
44	1,4435254982146673e-002
45	1,5869624927518540e-002
46	1,7456587420270394e-002
47	1,9196142460402233e-002
48	2,1118808557390057e-002
49	2,3224585711233862e-002
50	2,5543992431409649e-002
51	2,8107547227393413e-002
52	3,0915250099185155e-002
53	3,4028138065736867e-002
54	3,7415692617572556e-002
55	4,1169469283120215e-002
56	4,5258949552903834e-002
57	4,9806207464827418e-002
58	5,4780724509414958e-002
59	6,0274056215094456e-002
60	6,6286202581865905e-002
61	7,2908719138157288e-002
62	8,0202642902920618e-002
63	8,8229010895107887e-002
64	9,7048860133671075e-002
65	1,0675374614703818e-001
66	1,1743522446363720e-001
67	1,2918485061189611e-001
68	1,4209418012024294e-001
69	1,5628528702658162e-001
70	1,7191076387829218e-001
71	1,8912320322275461e-001
72	2,0804467909787286e-001
73	2,2882778405102694e-001
74	2,5171666615802485e-001
75	2,7689443647572254e-001
76	3,0457472457045198e-001
77	3,3503219702749720e-001
78	3,6854152043214211e-001
79	4,0537736136967073e-001
80	4,4593646046327096e-001
81	4,9052400280770286e-001
82	5,3956724753563035e-001
83	5,9352397228919340e-001
84	6,5288247322000792e-001
85	7,1816156498916595e-001
86	7,9000213629566329e-001
87	8,6898403881954400e-001
88	9,5590075380718409e-001
89	1,0514847254860074e+000
90	1,1566209906308176e+000
91	1,2722861415448470e+000
92	1,3995178075502792e+000
93	1,5394756920072024e+000
94	1,6934110538041323e+000
95	1,8627582628864405e+000
96	2,0490432447279274e+000
97	2,2539445173497725e+000
98	2,4793237098300120e+000
99	2,7272865993224893e+000
100	3,0000000000000000e+000

Table 9-55: StepSize table for 24-bit samples

0	3,5762791128491298e-006
1	3,9339070241340428e-006
2	4,4107442391805934e-006
3	4,7683721504655064e-006
4	5,2452093655120570e-006
5	5,8412558843202453e-006
6	6,4373024031284336e-006
7	7,0333489219366219e-006
8	7,7486047445064479e-006
9	8,4638605670762738e-006
10	9,4175349971693751e-006
11	1,0252000123500839e-005
12	1,1324883857355578e-005
13	1,2516976894971954e-005
14	1,3709069932588331e-005
15	1,5139581577727983e-005
16	1,6570093222867636e-005
17	1,8239023475530564e-005
18	2,0146372335716766e-005
19	2,2053721195902969e-005
20	2,4318697967374082e-005
21	2,6702884042606836e-005
22	2,9444698029124504e-005
23	3,2305721319403807e-005
24	3,5643581824729662e-005
25	3,9100651633817152e-005
26	4,3034558657951193e-005
27	4,7326093593370149e-005
28	5,2094465743835655e-005
29	5,7339675109347712e-005
30	6,3061721689906320e-005
31	6,9379814789273121e-005
32	7,6293954407448102e-005
33	8,3923349848192912e-005
34	9,2268001111507552e-005
35	1,0156632680491529e-004
36	1,1169911762465449e-004
37	1,2290479217824841e-004
38	1,3518335046569711e-004
39	1,4865400179076216e-004
40	1,6355516476096688e-004
41	1,7988683937631122e-004
42	1,9788744424431852e-004
43	2,1767618866875036e-004
44	2,3949149125713007e-004
45	2,6345256131321922e-004
46	2,8979781744454115e-004
47	3,1876567825861912e-004
48	3,5059456236297636e-004
49	3,8564209766889782e-004
50	4,2426591208766842e-004
51	4,6670442422681142e-004
52	5,1331526199761173e-004
53	5,6469447191887759e-004
54	6,2108047259813216e-004
55	6,8318851985794547e-004
56	7,5149545091336386e-004
57	8,2671652158695713e-004
58	9,0932856909377204e-004
59	1,0002852678639017e-003
60	1,1003018737199156e-003
61	1,2103320610919071e-003
62	1,3314487137137310e-003
63	1,4646055060154803e-003
64	1,6109945310347714e-003
65	1,7721655097205054e-003
66	1,9493105351102991e-003
67	2,1442177467605765e-003
68	2,3586752842277626e-003
69	2,5945904963720436e-003
70	2,8539899413573674e-003
71	3,1393770145627278e-003
72	3,4533743206708813e-003
73	3,7987236736683454e-003
74	4,1785245154529228e-003
75	4,5963531251374630e-003
76	5,0560242004423382e-003
77	5,5617100669992049e-003
78	6,1178214690472445e-003
79	6,7296036159519689e-003
80	7,4025401356864135e-003
81	8,1428299120461841e-003
82	8,9571486660419298e-003
83	9,8527681652031147e-003
84	1,0838033060793050e-002
85	1,1921884050593860e-002
86	1,3114096297513997e-002
87	1,4425517848195773e-002
88	1,5868069633015350e-002
89	1,7454864675386508e-002
90	1,9200327301064409e-002
91	2,1120431556753107e-002
92	2,3232462791498040e-002
93	2,5555613703204836e-002
94	2,8111222757246822e-002
95	3,0922297349250002e-002
96	3,4014586688826884e-002
97	3,7415985753057691e-002
98	4,1157608170224208e-002
99	4,5273428591898514e-002
100	4,9800759530157987e-002
101	5,4780847404104160e-002
102	6,0258872539862694e-002
103	6,6284783635709721e-002
104	7,2913297762071824e-002
105	8,0204615617348624e-002
106	8,8225017574431602e-002
107	9,7047578936526643e-002
108	1,0675228914645780e-001
109	1,1742748229831246e-001
110	1,2917031397465634e-001
111	1,4208735729305236e-001
112	1,5629603341770570e-001
113	1,7192568444319778e-001
114	1,8911816944100493e-001
115	2,0803001022696618e-001
116	2,2883310661710579e-001
117	2,5171640535788598e-001
118	2,7688804589367461e-001
119	3,0457679087839018e-001
120	3,3503452957088109e-001
121	3,6853794676517804e-001
122	4,0539174144169587e-001
123	4,4593089174400469e-001
124	4,9052399283933557e-001
125	5,3957635636047796e-001
126	5,9353406352210802e-001
127	6,5288742219059737e-001
128	7,1817609288407480e-001
129	7,8999373793527339e-001
130	8,6899314749159184e-001
131	9,5589243839889027e-001
132	1,0514817299225008e+000
133	1,1566298194682383e+000
134	1,2722928848615747e+000
135	1,3995221137430804e+000
136	1,5394743131964581e+000
137	1,6934218041207556e+000
138	1,8627639845328312e+000
139	2,0490403233814627e+000
140	2,2539444272451910e+000
141	2,4793389414952922e+000
142	2,7272728356448215e+000
143	2,9999998807906962e+000

9.9.9 Scaling coefficients lookup table

Table 9-56: DMixScaling_IndexTodB

0	-200 dB	8	-3,5 dB	16	-7,5 dB	24	-11,5 dB
1	0 dB	9	-4,0 dB	17	-8,0 dB	25	-12,0 dB
2	-0,5 dB	10	-4,5 dB	18	-8,5 dB	26	-12,5 dB
3	-1,0 dB	11	-5,0 dB	19	-9,0 dB	27	-13,0 dB
4	-1,5 dB	12	-5,5 dB	20	-9,5 dB	28	-13,5 dB
5	-2,0 dB	13	-6,0 dB	21	-10,0 dB	29	-14,0 dB
6	-2,5 dB	14	-6,5 dB	22	-10,5 dB	30	-14,5 dB
7	-3,0 dB	15	-7,0 dB	23	-11,0 dB	31	-15,0 dB

Table 9-57: DMixContribution_IndexTodB

0	-200 dB	11	-5,0 dB	22	-10,5 dB	33	-16,0 dB	44	-23,0 dB	56	-35,0 dB
1	0 dB	12	-5,5 dB	23	-11,0 dB	34	-16,5 dB	45	-24,0 dB	57	-36,0 dB
2	-0,5 dB	13	-6,0 dB	24	-11,5 dB	35	-17,0 dB	46	-25,0 dB	58	-37,0 dB
3	-1,0 dB	14	-6,5 dB	25	-12,0 dB	36	-17,5 dB	47	-26,0 dB	59	-38,0 dB
4	-1,5 dB	15	-7,0 dB	26	-12,5 dB	37	-18,0 dB	48	-27,0 dB	60	-39,0 dB
5	-2,0 dB	16	-7,5 dB	27	-13,0 dB	38	-18,5 dB	49	-28,0 dB	61	-40,0 dB
6	-2,5 dB	17	-8,0 dB	28	-13,5 dB	39	-19,0 dB	50	-29,0 dB		
7	-3,0 dB	18	-8,5 dB	29	-14,0 dB	40	-19,5 dB	51	-30,0 dB		
8	-3,5 dB	19	-9,0 dB	30	-14,5 dB	41	-20,0 dB	52	-31,0 dB		
9	-4,0 dB	20	-9,5 dB	31	-15,0 dB	42	-21,0 dB	53	-32,0 dB		
10	-4,5 dB	21	-10,0 dB	32	-15,5 dB	43	-22,0 dB	54	-33,0 dB		

9.9.10 Index Hopping Huffman Tables

Table 9-58: Codebook for Tonal Groups

Index	prsDist [nGroup]										tnlScf	
	0		1		2		3		4			
	A	B	A	B	A	B	A	B	A	B	A	B
0	0x01	0x00	0x01	0x00	0x01	0x00	0x01	0x00	0x01	0x00	0x01	0x00
1	0x01	0x38	0x01	0x30	0x01	0x38	0x01	0x36	0x01	0x2C	0x01	0x1A
2	0x01	0x04	0x01	0x16	0x01	0x1C	0x01	0x22	0x01	0x02	0x01	0x16
3	0x01	0x02	0x01	0x04	0x01	0x18	0x01	0x0C	0xFF	0x02	0x01	0x02
4	0xFF	0x05	0x01	0x02	0x01	0x16	0x01	0x0A	0x01	0x12	0xFF	0x03
5	0xFF	0x04	0xFF	0x04	0x01	0x14	0x01	0x04	0x01	0x10	0x01	0x12
6	0x01	0x26	0xFF	0x07	0x01	0x04	0x01	0x02	0x01	0x08	0x01	0x10
7	0x01	0x24	0x01	0x02	0x01	0x02	0xFF	0x08	0x01	0x06	0x01	0x02
8	0x01	0x22	0xFF	0x0A	0xFF	0x0E	0xFF	0x02	0x01	0x02	0xFF	0x0B
9	0x01	0x20	0x01	0x0E	0xFF	0x11	0x01	0x02	0xFF	0x07	0x01	0x0C
10	0x01	0x18	0x01	0x0C	0x01	0x02	0xFF	0x07	0x01	0x02	0x01	0x0A
11	0x01	0x02	0x01	0x0A	0xFF	0x0F	0x01	0x02	0xFF	0x0F	0x01	0x02
12	0xFF	0x13	0x01	0x08	0x01	0x0C	0xFF	0x17	0xFF	0x15	0xFF	0x10
13	0x01	0x14	0x01	0x04	0x01	0x02	0xFF	0x0C	0xFF	0x03	0x01	0x02
14	0x01	0x06	0x01	0x02	0xFF	0x17	0xFF	0x05	0x01	0x02	0xFF	0x11
15	0x01	0x04	0xFF	0x03	0x01	0x02	0x01	0x10	0xFF	0x06	0x01	0x02
16	0x01	0x02	0xFF	0x1B	0xFF	0x1C	0x01	0x0E	0x01	0x02	0xFF	0x12
17	0xFF	0x21	0x01	0x02	0x01	0x02	0x01	0x02	0xFF	0x0D	0x01	0x02
18	0xFF	0x1F	0xFF	0x1D	0xFF	0x1D	0xFF	0x0A	0x01	0x02	0xFF	0x13
19	0xFF	0x1C	0xFF	0x1C	0x01	0x04	0x01	0x0A	0xFF	0x0E	0xFF	0x00
20	0x01	0x0C	0xFF	0x16	0x01	0x02	0x01	0x02	0xFF	0x12	0xFF	0x0F
21	0x01	0x0A	0xFF	0x15	0xFF	0x1E	0xFF	0x14	0xFF	0x04	0xFF	0x0E
22	0x01	0x04	0xFF	0x0F	0xFF	0x00	0x01	0x02	0x01	0x08	0xFF	0x09
23	0x01	0x02	0xFF	0x0E	0xFF	0x1F	0xFF	0x19	0x01	0x02	0xFF	0x07

Index	prsDist [nGroup]										tnlScf	
	0		1		2		3		4			
	A	B	A	B	A	B	A	B	A	B	A	B
24	0xFF	0x22	0x01	0x18	0xFF	0x19	0x01	0x02	0xFF	0x05	0x01	0x02
25	0xFF	0x25	0x01	0x02	0xFF	0x0A	0xFF	0x1A	0x01	0x04	0xFF	0x02
26	0x01	0x04	0xFF	0x08	0xFF	0x08	0x01	0x02	0x01	0x02	0xFF	0x04
27	0x01	0x02	0x01	0x14	0x01	0x02	0xFF	0x1B	0xFF	0x0B	0x01	0x04
28	0xFF	0x23	0x01	0x02	0xFF	0x09	0xFF	0x00	0xFF	0x0A	0x01	0x02
29	0xFF	0x00	0xFF	0x10	0xFF	0x04	0xFF	0x16	0xFF	0x14	0xFF	0x01
30	0xFF	0x24	0x01	0x02	0x01	0x1A	0xFF	0x09	0x01	0x0C	0xFF	0x05
31	0xFF	0x20	0xFF	0x13	0x01	0x0E	0x01	0x04	0x01	0x0A	0x01	0x08
32	0xFF	0x1E	0x01	0x02	0x01	0x0C	0x01	0x02	0x01	0x08	0x01	0x06
33	0xFF	0x18	0xFF	0x17	0x01	0x0A	0xFF	0x0D	0x01	0x02	0x01	0x04
34	0x01	0x02	0x01	0x02	0x01	0x04	0xFF	0x11	0xFF	0x0C	0x01	0x02
35	0xFF	0x16	0xFF	0x1A	0x01	0x02	0xFF	0x04	0x01	0x02	0xFF	0x0C
36	0x01	0x02	0x01	0x02	0xFF	0x16	0x01	0x12	0xFF	0x10	0xFF	0x0D
37	0xFF	0x17	0xFF	0x1E	0xFF	0x03	0x01	0x0A	0x01	0x02	0xFF	0x0A
38	0x01	0x02	0x01	0x08	0x01	0x02	0x01	0x06	0xFF	0x16	0xFF	0x08
39	0xFF	0x1D	0x01	0x06	0xFF	0x15	0x01	0x02	0xFF	0x00	0xFF	0x06
40	0xFF	0x1B	0x01	0x02	0x01	0x02	0xFF	0x0E	0xFF	0x11		
41	0xFF	0x11	0xFF	0x21	0xFF	0x1A	0x01	0x02	0xFF	0x13		
42	0xFF	0x0E	0x01	0x02	0xFF	0x1B	0xFF	0x13	0x01	0x02		
43	0xFF	0x07	0xFF	0x22	0xFF	0x0C	0xFF	0x18	0xFF	0x08		
44	0x01	0x0C	0xFF	0x00	0xFF	0x0B	0x01	0x02	0xFF	0x09		
45	0x01	0x02	0xFF	0x20	0x01	0x0A	0xFF	0x03	0xFF	0x01		
46	0xFF	0x0C	0xFF	0x1F	0x01	0x04	0xFF	0x0B				
47	0x01	0x02	0xFF	0x0C	0x01	0x02	0x01	0x04				
48	0xFF	0x01	0xFF	0x05	0xFF	0x10	0x01	0x02				
49	0x01	0x06	0x01	0x0E	0xFF	0x12	0xFF	0x15				
50	0x01	0x04	0x01	0x02	0x01	0x04	0xFF	0x12				
51	0x01	0x02	0xFF	0x09	0x01	0x02	0x01	0x02				
52	0xFF	0x1A	0x01	0x02	0xFF	0x14	0xFF	0x10				
53	0xFF	0x03	0xFF	0x01	0xFF	0x18	0xFF	0x0F				
54	0xFF	0x19	0x01	0x08	0xFF	0x13	0xFF	0x06				
55	0xFF	0x14	0x01	0x06	0xFF	0x0D	0xFF	0x01				
56	0xFF	0x08	0x01	0x02	0xFF	0x05						
57	0x01	0x0E	0xFF	0x14	0x01	0x02						
58	0x01	0x04	0x01	0x02	0xFF	0x01						
59	0x01	0x02	0xFF	0x19	0x01	0x02						
60	0xFF	0x0A	0xFF	0x18	0xFF	0x06						
61	0xFF	0x0D	0xFF	0x12	0xFF	0x07						
62	0x01	0x08	0xFF	0x11								
63	0x01	0x04	0x01	0x02								
64	0x01	0x02	0xFF	0x06								
65	0xFF	0x0F	0x01	0x02								
66	0xFF	0x10	0xFF	0x0B								
67	0x01	0x02	0xFF	0x0D								
68	0xFF	0x12										
69	0xFF	0x15										
70	0xFF	0x0B										
71	0x01	0x02										
72	0xFF	0x09										
73	0xFF	0x06										

Table 9-59: Codebook for Amplitude and Phase

Index	rsdAmp		fstRsdAmp		dPH		dAmp		rsdAmp	
	A	B	A	B	A	B	A	B	A	B
0	0x01	0x00	0x01	0x00	0x01	0x00	0x01	0x00	0x01	0x00
1	0x01	0x20	0x01	0x26	0x01	0x0C	0x01	0x0C	0x01	0x02
2	0x01	0x04	0x01	0x1C	0x01	0x02	0x01	0x02	0xFF	0x01
3	0x01	0x02	0x01	0x04	0xFF	0x02	0xFF	0x02	0x01	0x02
4	0xFF	0x02	0x01	0x02	0x01	0x02	0x01	0x02	0xFF	0x02
5	0xFF	0x01	0xFF	0x0C	0xFF	0x03	0xFF	0x03	0x01	0x02
6	0x01	0x02	0xFF	0x11	0x01	0x02	0x01	0x02	0xFF	0x03
7	0xFF	0x05	0x01	0x06	0xFF	0x04	0xFF	0x04	0x01	0x02
8	0x01	0x18	0x01	0x04	0x01	0x02	0x01	0x02	0xFF	0x04
9	0x01	0x16	0x01	0x02	0xFF	0x05	0xFF	0x05	0x01	0x02
10	0x01	0x12	0xFF	0x01	0x01	0x02	0x01	0x02	0xFF	0x05
11	0x01	0x0C	0xFF	0x08	0xFF	0x06	0xFF	0x06	0xFF	0x00
12	0x01	0x02	0xFF	0x09	0xFF	0x00	0xFF	0x00		
13	0xFF	0x0E	0x01	0x06	0xFF	0x01	0xFF	0x01		
14	0x01	0x02	0x01	0x04	0xFF	0x05				
15	0xFF	0x14	0x01	0x02	0xFF	0x00				
16	0x01	0x02	0xFF	0x14	0xFF	0x04				
17	0xFF	0x1A	0xFF	0x03	0xFF	0x08				
18	0x01	0x04	0xFF	0x05						
19	0x01	0x02	0x01	0x02						
20	0xFF	0x19	0xFF	0x06						
21	0xFF	0x20	0x01	0x02						
22	0xFF	0x13	0xFF	0x02						
23	0x01	0x02	0x01	0x06						
24	0xFF	0x10	0x01	0x02						
25	0x01	0x02	0xFF	0x16						
26	0xFF	0x18	0x01	0x02						
27	0xFF	0x11	0xFF	0x17						
28	0x01	0x02	0xFF	0x00						
29	0xFF	0x0C	0xFF	0x15						
30	0xFF	0x0D	0x01	0x08						
31	0xFF	0x09	0x01	0x02						
32	0xFF	0x07	0xFF	0x0B						
33	0x01	0x02	0x01	0x02						
34	0xFF	0x03	0xFF	0x13						
35	0x01	0x02	0x01	0x02						
36	0xFF	0x04	0xFF	0x07						
37	0x01	0x1C	0xFF	0x04						
38	0x01	0x04	0xFF	0x10						
39	0x01	0x02	0x01	0x06						
40	0xFF	0x08	0x01	0x04						
41	0xFF	0x0B	0x01	0x02						
42	0x01	0x16	0xFF	0x0A						
43	0x01	0x04	0xFF	0x12						
44	0x01	0x02	0xFF	0x0F						
45	0xFF	0x12	0x01	0x02						
46	0xFF	0x0F	0xFF	0x0D						
47	0x01	0x10	0xFF	0x0E						
48	0x01	0x0C								
49	0x01	0x0A								
50	0x01	0x02								
51	0xFF	0x1E								
52	0x01	0x04								
53	0x01	0x02								
54	0xFF	0x24								
55	0xFF	0x22								
56	0x01	0x02								
57	0xFF	0x1D								

Index	rsdAmp		fstRsdAmp		dPH		dAmp		rsdAmp	
	A	B	A	B	A	B	A	B	A	B
58	0xFF	0x00								
59	0xFF	0x15								
60	0x01	0x02								
61	0xFF	0x1C								
62	0xFF	0x17								
63	0xFF	0x16								
64	0xFF	0x0A								
65	0xFF	0x06								

Table 9-60: Code book for Grid Reconstruction

Index	stGrid		avgG3	
	A	B	A	A
0	0x01	0x00	0x01	0x00
1	0x01	0x2A	0x01	0x20
2	0x01	0x28	0x01	0x1E
3	0x01	0x04	0x01	0x0A
4	0x01	0x02	0x01	0x02
5	0xFF	0x04	0xFF	0x0E
6	0xFF	0x03	0x01	0x06
7	0x01	0x02	0x01	0x02
8	0xFF	0x08	0xFF	0x0B
9	0x01	0x20	0x01	0x02
10	0x01	0x0A	0xFF	0x13
11	0x01	0x08	0xFF	0x09
12	0x01	0x02	0xFF	0x0D
13	0xFF	0x0E	0x01	0x12
14	0x01	0x02	0x01	0x10
15	0xFF	0x07	0x01	0x02
16	0x01	0x02	0xFF	0x0A
17	0xFF	0x09	0x01	0x04
18	0xFF	0x16	0x01	0x02
19	0xFF	0x0C	0xFF	0x14
20	0x01	0x14	0xFF	0x08
21	0x01	0x0E	0x01	0x08
22	0x01	0x02	0x01	0x06
23	0xFF	0x10	0x01	0x02
24	0x01	0x02	0xFF	0x06
25	0xFF	0x0B	0x01	0x02
26	0x01	0x02	0xFF	0x17
27	0xFF	0x0D	0xFF	0x00
28	0x01	0x04	0xFF	0x15
29	0x01	0x02	0xFF	0x07
30	0xFF	0x11	0xFF	0x0C
31	0xFF	0x0F	0xFF	0x12
32	0x01	0x02	0xFF	0x10
33	0xFF	0x18	0x01	0x02
34	0xFF	0x00	0xFF	0x0F
35	0x01	0x02	0xFF	0x11
36	0xFF	0x12		
37	0x01	0x02		
38	0xFF	0x14		
39	0xFF	0x02		
40	0xFF	0x05		
41	0xFF	0x0A		
42	0xFF	0x01		
43	0xFF	0x06		

Annex A (informative): Bibliography

Zoran Fejzo: "DTS Coherent Acoustics; Core and Extensions, Overview of Technology and Description of DTS Stream Frame Headers".

DTS, Inc. (5220 Las Virgenes Rd., Calabasas, CA 91302): "DTS Coherent Acoustics Core" (DTS Document #9302F33500).

DTS, Inc. (5220 Las Virgenes Rd., Calabasas, CA 91302): "Coherent Acoustics Extensions" (DTS Document #9302F41300).

DTS, Inc. (5220 Las Virgenes Rd., Calabasas, CA 91302): "DTS-HD Substream and Decoder Interface Description" (DTS Document No.: 9302F30400).

DTS, Inc. (5220 Las Virgenes Rd., Calabasas, CA 91302): "DTS-HD Lossless Extension" (DTS Document #9301E96800).

DTS, Inc. (5220 Las Virgenes Rd., Calabasas, CA 91302): "DTS LBR Bitstream Format Specification"(DTS Document #9302F27600).

IETF RFC 2119: "Key words for use in RFCs to Indicate Requirement Levels", S. Bradner, March 1997.

NOTE: Available at http://www.ietf.org/rfc/rfc2119.txt.

ISO 639-3: "Codes for the representation of names of language - Part 3: Alpha-3 code for comprehensive coverage of languages".

NOTE: Available at International Standards Organization, www.iso.ch; International Electrotechnical Commission, www.iec.ch.

Annex B (normative): CRC Algorithm

A number of CRC algorithms are in use throughout the world. This Annex described the CRC method used by all of the DTS algorithms. This algorithm of CRC is also known as CRC-CCITT.

The CRC is used for two purposes: to verify the correctness of header data and to greatly reduce the probability of false-alias sync-word detection. When flagged, a CRC checksum is added to each packet within the stream. CRCs are used to verify the data of various components within the frame. It is important to verify the header information, since the mechanism for traversing through the frame is by reading the size of a component from the NAVI table and locating the start position of the component based on its value.

The CRC16 polynomial is:

$$G(x) = x^{16} + x^{12} + x^5 + 1$$

The CRC16 is initialized to the value of *0xFFFF* before checksum computation commences.

Annex C (informative): Example Pseudocode

C.1 About Annex C

This annex provides detail on various processing algorithms that will assist the reader by providing a top level view of critical processes, as well as examples to aid in correctly interpreting the bitstream.

C.2 Overview of main function calls

This clause outlines in detail pseudocode examples to clarify the details of the main function calls, unpacking of the frame and primary audio coding headers.

Based on this subframe structure, the procedure of decoding a subframe may be illustrated by the following pseudocode:

```
DecodeSubframe() {

// Unpack Side Information.
UnpackSideInformation();

// Inverse VQ to extract high frequency subbands.
for (nChannel=0; nChannel<nNumPrimaryChannels; nChannel++) {
    for (nSubband=nHFreqVQBegin; nSubband<nHFreqVQEnd; nSubband++) {
        VQIndex = ExtractVQIndex();
        InverseVQ(VQIndex);     // One index looks up 32 samples in one subband analysis window.
}   }

// Unpack the LFE channel
ExtractLFEDecimatedSamples();   // Extract the decimated samples.
InterpolateLFESamples();            // Interpolate for all LFE samples.

// Unpack subsubframes.
for (nSubsubframe=0; nSubsubframe<nNumOfSubsubframes; nSubsubframe++) {
    UnpackSubsubframe();
}

// Reconstruct all primary channels through filter bank interpolation
for (nChannel=0; nChannel<nNumPrimaryChannels; nChannel++) {
    ReconstructChannel();
}}
```

A subsubframe consists of eight subband samples (a subband analysis subwindow) for each subband of all primary channels, so its decoding procedure may be described as:

```
UnpackSubsubframe() {
    for (nChannel=0; nChannel<nNumPrimaryChannels; nChannel++)
        for (nSubband=0; nSubband<nHFreqVQBegin; nSubband++)
            UnpackOneSubwindow(); // Get 8 subband samples.
}
```

An example of synchronization and decoding procedure may be described as follows:

```
START_SYNC:         InSyncFlag = 0;     // Search for extend sync word (38-bit sync word +
extension)
SearchForExtSync();

// Search for another sync word (32-bit sync word)
SearchForSync();

// Count the distance between the two sync words and check if it is within the
// limits. The next sync word is expected at this distance.
InSyncFlag = CountSyncDist();
    if (InSyncFlag==1)
        DecodeOneFrame();       // Decode the received frame
    else
        Goto START_SYNC;        // Decode the remaining frames
while (NotEndOfBitStream) {     // Check if sync word occurred at the expected interval
    InSyncFlag = CheckSync();
```

```
    if (InSyncFlag==1)
        DecodeOneFrame();
    else
        Goto START_SYNC;
}
```

C.3 Decoding Algorithms

C.3.1 About Decoding Algorithms

This clause outlines the decoding routines utilized by Coherent Acoustics.

C.3.2 Block Code

Two versions of the block code decoder are presented here based on:

- the table look-up method;
- the arithmetic method that requires one modulus division and one integer division per one decoded quantization index.

The table look-up based decoding of a block code may be best illustrated by an example. Suppose a code of 64 is received as a three level block code. This code can be decoded as follows:

1^{st} Element: $64 = 3 \times 21 + 1$; so quantization index = 0

2^{nd} Element: $21 = 3 \times 7 + 0$; so quantization index = -1

3^{rd} Element: $7 = 3 \times 2 + 1$; so quantization index = 0

4^{th} Element: $2 = 3 \times 0 + 2$; so quantization index = +1

where the quantization indexes are obtained by using the residuals to look up the quantization index table [-1, 0, 1]. In summary, the quantization indexes of the four samples are (0, -1, 0, +1).

The same code can be decoded using the code book of table V.3 in clause D.6.1. In order to facilitate the decoding process, this table is rearranged to give Table C-1. Then this code of 64 is decoded as follows:

4^{th} Element: $64 - 54 = 10 \geq 0$; so quantization index = +1

3^{rd} Element: $10 - 9 = 1 \geq 0$; so quantization index = 0

2^{nd} Element: $1 - 0 = 1 \geq 0$; so quantization index = -1

1^{st} Element: $1 - 1 = 0 \geq 0$; so quantization index = 0

Therefore, the quantization indexes of the four samples are (0, -1, 0, +1). A general decoding procedure is given in the following pseudocode, assuming that the block codes in clause D.6 are rearranged as in Table C-1.

Table C-1: 3-level 4-element 7-bit Block Code Book

	Quantization Level index	-1	0	+0
Code	1st Element	0	1	2
For	2nd Element 0 3 6	0	3	6
	3rd Element 0 9 18	0	9	18
	4th Element 0 27 54	0	27	54

```
int DecodeBlockCode(int nCode, int *pnValue) {
    // nCode:   Input code to be decoded.
    // nNumElement:     Number of elements (samples) encoded
    // in a block.
```

```
    // nNumLevel:   Number of quantization levels.
    // *pnValue:    Array of decoded sample values.
    // *pnTable:    Pointer to the code book. The code book is
    //  organized as an array, each row of which contains
    //  the code book for a particular element (sample).
    pnValue += 3;
    nOffset = (nNumLevel-1)/2;
    int *pnEntry; // Pointer to the entries in the code book.
    for (int n=nNumElement; n>0; n--) {
        pnEntry = pnTable + n*nNumLevel; // Point to the last entry

        // in the code book.
        for (int m=0; m<nNumLevel; m++) {
            pnEntry--;
            if ( nCode >= *pnEntry ) {
                nCode -= *pnEntry;
                *pnValue = nOffset-m; // quantization index is calculated.
                if ( nCode<0 ) {
                    printf("ERROR: block code look-up fail.\n");
                    return NULL;
                }
                break;
            }
        }
        pnValue--;
    }

    // Check if look-up successful
    if ( nCode == 0 )
        return 1;
    else {
        printf("ERROR: block code lock-up fail.\n");
        return NULL;
    }
}
```

Very compact version of the block code decoder that does not use table look-up can be obtained using the modulus and integer division. The pseudocode that implements this version of the decoder is listed below:

```
int DecodeBlockCode(int nCode, int *pnValue) {
     // nCode:  Input code to be decoded.
    // nNumElement:     Number of elements (samples) encoded in a block.
    // nNumLevel:   Number of quantization levels.
    // *pnValue:    Array of decoded sample values.
    nOffset = (nNumLevel-1)>>1;
    for (int n=0; n< nNumElement; n++) {
        pnValue[n] = (nCode % nNumLevel) - nOffset;
        nCode /= nNumLevel;
    }
    if ( nCode == 0 )
        return 1;
    else {
        printf("ERROR: block code lock-up fail.\n");
        return NULL;
    }
}
```

C.3.3 Inverse ADPCM

Inverse ADPCM process is executed for each sample in a subband whose PMODE=1:

```
void InverseADPCM(void) {
// NumADPCMCoeff =4, the number of ADPCM coefficients.
// raADPCMcoeff[] are the ADPCM coefficients extracted
// from the bit stream.
// raSample[NumADPCMCoeff], ..., raSample[-1] are the
// history from last subframe or subsubframe. It is
// updated each time before reverse ADPCM is run for a
// block of samples for each subband.
for (m=0; m<nNumSample; m++)
    for (n=0; n<NumADPCMCoeff; n++)
        raSample[m] += raADPCMcoeff[n]*raSample[m-n-1];
}
```

C.3.4 Joint Subband Coding

```
for (ch=0; ch<nPCHS; ch++)
    if ( JOINX[ch]>0 ){            // Joint subband coding enabled.
        nSourceCh = JOINX[ch]-1;    // Get source channel. JOINX counts

        // channels as 1,2,3,4,5, so minus 1.
        for (n=nSUBS[ch]; n<nSUBS[nSourceCh]; n++)
            for (nSample=0; n<8*nSSC; nSample++)
                aPrmCh[ch].aSubband[n].aSample[nSample] = JOIN_SCALES[ch][n] *
aPrmCh[nSourceCh].aSubband[n].aSample[nSample];
    }
}
```

C.3.5 Sum/Difference Decoding

If flag SUMF is set, the front left and right channels are sum/difference encoded and are therefore appropriately decoded to produce the correct signals for the front left and right channels. Decoding is achieved by operating on the reconstructed subband samples:

```
for (n=0; n<nSUBS; n++) // All active subbands.
    for (nSample=0; nSample<8*nSSC; nSample++) { // Samples in all subsubframes
        FrontLeft[nSample] = Fleft[nSample] + Fright[nSample];
        Frontright[nSample] = Fleft[nSample] - Fright[nSample];
    }
```

This decoding is also required when AMODE = 3.

Similarly when SUMS is set the reconstructed subband samples of the Left and right surround channels are decoded as:

```
    for (n=0; n<nSUBS; n++) // All active subbands.
        for (nSample=0; nSample<8*nSSC; nSample++) { // Samples in all subsubframes
            SurroundLeft[nSample] = Sleft[nSample] + Sright[nSample];
            Surroundright[nSample] = Sleft[nSample] - Sright[nSample];
        }
```

C.3.6 Filter Bank Reconstruction

Having prepared all the subband samples, it is time to go through subband interpolation to reconstruct the PCM samples for each primary channel. As discussed before, there are two filter banks, one for perfect reconstruction and the other for non-perfect. The encoder indicates its choice to the decoder through the FILTS flag in the frame header.

```
for (ch=0; ch<nPCHS; ch++)
    aPrmCh[ch].QMFInterpolation(FILTS, nSUBS[ch]);

// FILTS indicates which filter bank to use
// nSUBS[ch] indicates the number of active subbands. Subbands
// above it are all zeros. For joint intensity coded subbands,
// it is set to that of the source channel, in order to
// reflect the true subband activity.
```

There are many methods to efficiently implement the reconstruction filter bank. Presented here is one possible solution. The two sets of 512 FIR coefficients are tabulated in clause D.8 to include both perfect reconstruction and nonperfect reconstruction and the selection is flagged by FILTS in the frame header.

The first step is to pre-calculate the cosine modulation coefficients:

```
PreCalCosMod() {
    for (j=0,k=0;k<16;k++)
        for (i=0;i<16;i++)
            raCosMod[j++] = (real)cos((2*i+1)*(2*k+1)*Pi/64);
    for (k=0;k<16;k++)
        for (i=0;i<16;i++)
            raCosMod[j++] = (real)cos((i)*(2 * k + 1)*Pi / 32);
    for (k=0;k<16;k++)
        raCosMod[j++] = real(0.25/(2 * cos((2 * k + 1)*Pi / 128)));
    for (k=0;k<16;k++)
        raCosMod[j++] = real(-0.25/(2 * sin((2 * k + 1)*Pi / 128)));
}
```

The filter bank reconstruction is illustrated by the following pseudocode:

```
QMFInterpolation(FILTS, int nSUBS) {
    // Select filter
    if ( FILTS==0 )             // Non-perfect reconstruction
        prCoeff = raCoeffLossy;
    else // Perfect reconstruction
        prCoeff = raCoeffLossLess;

    // Interpolation begins
    nChIndex = 0;   // Reconstructed channel sample index
    for (nSubIndex=nStart; nSubIndex<nEnd; nSubIndex++) {   // Subband samples

        // Load in one sample from each subband
        for (i=0; i<nSUBS; i++)
            raXin[i] = aSubband[i].raSample[nSubIndex];
        for (i=nSUBS; i<NumSubband; i++) // Clear inactive subbands
            raXin[i] = 0.0;

        //Multiply by cosine modulation coefficients and
        // Create temporary arrays SUM and DIFF.
        for (j=0,k=0;k<16;k++) {
            A[k] = (real)0.0;
            for (i=0;i<16;i++)
                A[k]+=(raXin[2*i]+raXin[2*i+1])*raCosMod[j++];
        }
        for (k=0;k<16;k++) {
            B[k] = (real)0.0;
            for (i=0;i<16;i++) {
                if(i>0)
                    B[k]+=(raXin[2*i]+raXin[2*i-1])*raCosMod[j++];
                else
                    B[k]+=(raXin[2*i])*raCosMod[j++];
            }
            SUM[k]=A[k]+B[k];
            DIFF[k]=A[k]-B[k];
        }

        // Store history
        for (k=0;k<16;k++)
            raX[k]=raCosMod[j++]*SUM[k];
        for (k=0;k<16;k++)
            raX[32-k-1]=raCosMod[j++]*DIFF[k];

        // Multiply by filter coefficients
        for(k=31,i=0;i<32;i++,k--)
            for(j=0;j<512;j+=64)
                raZ[i] += prCoeff[i+j]*(raX[i+j]-raX[j+k]);
        for(k=31,i=0;i<32;i++,k--)
            for(j=0;j<512;j+=64)
                raZ[32+i] += prCoeff[32+i+j]*(-raX[i+j]-raX[j+k]);

        // Create 32 PCM output samples
        for(i=0;i<32;i++)
            naCh[nChIndex++] = int(rScale*raZ[i]);

        // Update working arrays
        for(i=511;i>=32;i--)
            raX[i] = raX[i-32];
        for(i=0;i<NumSubband;i++)
            raZ[i] = raZ[i+32];
        for(i=0;i<NumSubband;i++)
            raZ[i+32] = (real)0.0;
    }
}
```

C.3.7 Interpolation of LFE Channel

```
void InterpolationFIR(int nDecimationSelect) {
    // rLFE:    An array holding decimated samples.
    //  Samples in current subframe starts from rLFE[0],
    //  while rLFE[-1], rLFE[-2], ..., stores samples
    //  from last subframe as history.
    // naCh:    An array holding interpolated samples
    // Select decimation filter
    if (nDecimationSelect==1) {// 128 decimation
```

```
        nDeciFactor = 128;  // Decimation factor = 128
        prCoeff = raCoeff128;   // Point to the 128X FIR coefficient array
    }
    else {  // 64 decimation
        nDeciFactor = 64;
        prCoeff = raCoeff64;
    }
    // Interpolation
    NumFIRCoef = 512; // Number of FIR coefficients
    nInterpIndex = 0; // Index to the interpolated samples
    for (nDeciIndex=0; nDeciIndex<nNumDeciSample; nDeciIndex++) {
        // One decimated sample generates nDeciFactor interpolated ones.
        for (k=0; k<nDeciFactor; k++) {
            // Clear accumulation
            rTmp = 0.0;
            // Accumulate
            for (J=0; J<NumFIRCoef/nDeciFactor; J++)
                rTmp += rLFE[nDeciIndex-J]*prCoeff[k+J*nDeciFactor];
            // Save interpolated samples as integer
            naCh[nInterpIndex++] = (int)rTmp;
        }
        nDeciIndex++; // Next decimated sample
    }
}
```

C.4 Coefficients for Remapping Loudspeaker Locations

The coefficients that control the remapping of loudspeaker locations are transmitted using the 5 bit codes, each corresponding to an index into a scale factor lookup table.

The range of coefficients is between -15 dB and 0 dB in steps of 0,5 dB. A subset of the Scale Factor Table (see clause D.11) is used to map the 5-bit codes to 16-bit fixed point values. The table entries are unsigned 16 bit integer numbers representing the numbers in column AbsValues of the same table, after multiplication by 2^{15} and rounding to the nearest integer value.

The coefficients (SpkrRemapCoeff) are obtained from the transmitted 5 bit codes (SpkrRemapCodes) in the following manner:

```
TblIndex =  (nuSpkrRemapCodes << 2) + 120;
if (TblIndex > 240){
    Error: Invalid Index For a Speaker Remapping Coefficient
}
SpkrRemapCoeff = LinScalesTable[TblIndex];
```

C.5 Post Mix Gain Adjustment

The scale factors that adjust the gain in all channels after combining audio assets are transmitted using the 6 bit codes, each corresponding to an index into a scale factor lookup table. The range of scale factors is between -15 dB and 15 dB in steps of 0,5 dB.

The Scale Factor Table (see clause D.11) is the lookup table used to map the 6-bit codes to values available in the LinScalesTable column. The table entries are unsigned 16-bit integers representing the numbers in column AbsValues of the same table, after multiplication by 2^{15} and rounding to the nearest integer value.

The coefficient (PostMixGainAdj) is obtained from the transmitted 6-bit code (nuPostMixGainAdjCode) in the following manner:

```
if (nuPostMixGainAdjCode > 60){
    Error: Invalid Index For a Post Mix Gain Adjustment Code
}
if (nuPostMixGainAdjCode == 30)

    // 0 dB gain adjustment
    PostMixGainAdj = 1 << 20;
else
{
    // Look-up linear scale corresponding to range from -60 dB to -30 dB
    PostMixGainAdj = LinScalesTable[nuPostMixGainAdjCode];
```

```
    // Translate from range [-60 dB, -30 dB] to [-15 dB, +15 dB]
    // 5827066 (Q15) -> 177.8279 -> +45 dB
    // Also translate the PostMixGainAdj from 16-bit in Q15 to 24-bit in Q20
    PostMixGainAdj = (PostMixGainAdj *5827066 + (1 << 9)) >> 10;
}
```

The resulting coefficient PostMixGainAdj is a 24-bit unsigned fixed-point number in Q20 format.

C.6 Coefficients for Mixing Audio Assets

The coefficients that control the mixing audio assets are transmitted using the 6-bit codes, each corresponding to an index into a scale factor lookup table.

The range of scale factors is between -60 dB and 0 dB with addition of -∞ (ScaleFactor = 0) which is coded with a code 0. Furthermore the range [-60 to 0] is subdivided into 3 regions each with different grid resolution:

1) [-60 to -30] with resolution of 2 dB

2) [-29 to -15] with resolution of 1 dB

3) [-14,5 to 0] with resolution of 0,5 dB

A subset of the Scale Factor Table (see clause D.11) is used to correlate the index to the integer scalars. The table entries are unsigned 16 bit integers representing the numbers in column AbsValues of the same table, after multiplication by 2^{15} and rounding to the nearest integer value.

The coefficients (ScaleFactors) are obtained from the transmitted 6-bit codes (CoeffCodes) in the following manner:

```
if (CoeffCodes == 0)
    ScaleFactors = 0
else {
    ScaleFactorsTblIndex = (CoeffCodes - 1) << 2;
    if (ScaleFactorsTblIndex > 240)
        Error: Invalid Index For a Scale Factor
    ScaleFactors = LinScalesTable[ScaleFactorsTblIndex];
}
```

C.7 Smoothing the Coefficient Transitions

The downmix coefficients, mixing and scaling coefficients may change their values from frame to frame. In order to ensure a smooth transition, the coefficient interpolation is performed over the frame that carries the new values for the coefficients. The interpolation is performed on all coefficients in a similar manner. In particular the coefficient linear interpolation is performed over the nFrmSize sample intervals. The value of actual coefficient Coeff(n) to be used at the time instance n (n = 0 corresponds to the first sample in the current frame) is obtained using the procedure outlined below:

$Delta_k$ = $ScaleFactors_k$ - $ScaleFactors_{k-1}$;
nShift = $\log_2$(nFrmSize); // nFrmSize = 2^{nShift}

```
if ( |Delta_k| > 0 ){
    Ramp = 0; // Ramp needs to be 32-bit variable
    for (n=0; n<nFrmSize; n++){
        Coeff[n++] = ScaleFactors_{k-1} + (Ramp + (1<<(nShift-1)))>>nShift;
        Ramp += Delta_k;
    }
}
else{
    for (n=0; n<nFrmSize; n++)
        Coeff[n] = ScaleFactors_k;
}
```

where the >> indicates the right shift operation, the << indicates the left shift operation and the $ScaleFactors_k$ are the coefficients transmitted in the stream in the frame k.

The resulting values of Coeff(n) are in the same fixed-point number format as the original $ScaleFactors_k$. The new coefficient value ($ScaleFactors_k$) is reached over the period that corresponds to nFrmSize samples.

C.8 Entropy Coding

Entropy coding removes redundancy from the residual signal *e(n)* without any loss of information. In DTS-HD lossless codec, two types of codes-Rice codes and Binary codes-are used. Both codes are simple to implement on the encoder and the decoder side. Binary codes are optimal codes for a uniform distribution of numbers that are to be coded. On the other hand, the Rice codes are Huffman codes for a Laplacian distribution of numbers that need to be coded. As it turns out, the distribution of the most of the residual signals obtained after linear prediction are close to the Laplacian. Highly uncorrelated input audio signals are an exception; their residual distribution is closer to uniform.

Binary code is characterized by a single parameter BitWidth that represents the number of bits used for each code word. The code word is just a binary representation of unsigned integer numbers in the range between 0 and $2^{BitWidth} - 1$.

The Rice codes are also characterized by a single parameter, denoted as *KRice*. The Rice code with parameter *KRice* for an unsigned integer valued residual e(n) is constructed from two parts:

1) A unary representation of {*u(n)>>KRice* } where >> denotes a right shift operation and a unary representation of number k consisting of k "0" bits followed by a single stop bit "1" (i.e. unary representation of 5 is 000001).

2) *KRice* least significant bits of e(n), i.e. *KRice* bit binary representation *of {u(n) & (1<< KRice -1)}* where << denotes a left shift operation and & denotes a bitwise AND operation.

The above definitions for both Binary and Rice codes assume coding of the unsigned integer numbers. Audio samples, as well as the prediction residual samples (both denoted by e(n)) in our codec, are signed integers and prior to the entropy coding, these samples need to be translated to the unsigned integer representation u(n) using the following mapping:

$$u(n) = \begin{cases} e(n) << 1, & \forall e(n) \geq 0 \\ ((-e(n)) << 1) - 1, & \forall e(n) < 0 \end{cases}$$

where << denotes the left shift operation.

The coding parameter for both types of codes is determined for the duration of one segment. The choice of the code (Binary or Rice) is also determined for one segment.

In a channel set, the code selection (bRiceCodeFlag) and the corresponding coding parameter (ncABIT) can be chosen:

1) For each channel separately (ncSegType = 0) i.e. all samples in the segment of channel 1 are coded using (bRiceCodeFlag1, ncABIT1), all samples in the segment of channel 2 are coded using (bRiceCodeFlag2, ncABIT2).

2) For all channels jointly (ncSegType = 1) i.e. all samples in the segment for all channels in the channel set are coded using single bRiceCodeFlag and single ncABIT.

As indicated above, the selection between 1) and 2) is carried in the parameter ncSegType.

For one channel set in one segment, the following components are packed:

1) ncSegType

2) bRiceCodeFlag[] (1 per channel when ncSegType=0 or single for all channels when ncSegType =1)

3) ncABIT[] (1 per channel when ncSegType=0 or single for all channels when ncSegType =1)

4) the entropy codes for all samples in one segment of channel 1

5) the entropy codes for all samples in one segment of channel 2

6) the entropy codes for all samples in one segment of channel 3

On the decode side the entropy codes are extracted, as unsigned numbers u(n), according to the previously extracted coding parameters (ncSegType, bRiceCodeFlag[] and ncABIT[]).

To reconstruct signed integer representation of residual samples e(n) from the extracted unsigned integer numbers u(n), the following mapping is performed:

$$e(n) = \begin{cases} u(n) >> 1, & \forall \ even\ numbered\ u(n) \\ -(u(n) >> 1) - 1, & \forall \ odd\ numbered\ u(n) \end{cases}$$

C.9 Downmix Coefficients

The remainder of this text defines the following parameters:

1) DmixCoeff as the fixed-point representation for entries of matrix *Dmix_Mtrx*

2) DmixScale as the fixed-point representation for entries of matrix *Scale_Mtrx*

3) *InvDmixScale* as the fixed-point representation for entries of matrix *InvScale_Mtrx*

Both the DmixCoeff and DmixScale parameters are transmitted using 9-bit codes DmixCoeffCodes and DmixScaleCodes respectively. Each consists of a sign and an index for the table lookup of its absolute value |DmixCoeff| or |DmixScale|.

Since both the DmixScale and the InvDmixScale are needed on the decode side. there are two look-up tables, as defined in clause D.11:

- DmixTable[ ] is used for finding the absolute value of both DmixCoeff and DmixScale parameters
- InvDmixTable[ ] is used for finding the absolute value of InvDmixScale parameter

Note that the range of the entries in DmixTable[ ] is between -60 dB and 0 dB with addition of -∞ (|DMixCoeff| =0), which is coded with a DmixCode=0. Furthermore, the range [-60 to 0] is subdivided into 3 regions, each with a different grid resolution:

1) [-60 to -30] with resolution of 0,5 dB

2) [-29,75 to -15] with resolution of 0,25 dB

3) [-14,875 to 0] with resolution of 0,125 dB

Although the DmixScale parameters are obtained from the DmixTable[ ], their range is limited, on the encode side, to [-40 dB, 0 dB]. Consequently the InvDmixTable[ ], the look-up table for the InvDmixScale parameters, has entries that correspond to the inverse of |DmixScale| over the limited range [-40 dB to 0 dB].

The entries in DmixTable column of clause D.11 are unsigned 16-bit integer numbers representing the numbers in column AbsValues of clause D.11, after multiplication by 2^{15} and rounding to the nearest integer value. The DMixCoeff are signed integer numbers obtained from the entries of DmixTable after multiplication by the sign value DmixSign.

The DmixCoeff parameters are obtained from the transmitted 9-bit codes DmixCoeffCodes in the following manner:

1) Extract Sign Bit: the most significant bit represents the sign bit such that:

   - (DmixCoeffCodes and 0x100)>>8 = 1 → DmixSign = 1
   - (DmixCoeffCodes and 0x100)>>8 = 0 → DmixSign = -1

2) Look up the |DmixCoeff|: the lower 8 bits of the DmixCoeffCodes represent the index into the DmixTable []; the DMixCoeff parameters are calculated in the following manner:

```
if ((DmixCoeffCodes & 0xFF) == 0)
    DMixCoeff = 0;
else {
    DmixTblIndex = (DmixCoeffCodes & 0xFF) - 1;
```

```
    if (DmixTblIndex > 240)
        Error: Invalid Index for a Downmix Coefficient

    DMixCoeff = DmixSign *DmixTable[DmixTblIndex];
}
```

The DmixScale parameters are obtained from the transmitted 9-bit codes DmixScaleCodes in the following manner:

1) 1) Extract Sign Bit: the most significant bit represents the sign bit such that:

   - (DmixScaleCodes and 0x100)>>8 = 1 → DmixScaleSign = 1

   - (DmixScaleCodes and 0x100)>>8 = 0 → DmixScaleSign = -1

2) Look up the |DmixScale|: the lower 8 bits of the DmixScaleCodes represent the index into the DmixTable[ ];

   a) the DmixScale parameters are calculated in the following manner:

```
DmixScaleTblIndex = (DmixScaleCodes & 0xFF) - 1
If ((DmixScaleTblIndex < 40)  || (DmixScaleTblIndex >240) )
Error: Invalid Index for a Downmix Scaling Parameter
DmixScale = DmixScaleSign × DmixTable[DmixScaleTblIndex];
```

The entries in column InvDmixTable of clause D.11 are unsigned 24-bit integer numbers representing the numbers from column InvAbsValues of clause D.11, after multiplication by 2^{16} and rounding to the nearest integer value. The InvDmixScale are signed 24-bit integer numbers obtained from the entries of InvDmixTable[ ] after multiplication by the sign value DmixScaleSign.

The InvDmixScale parameters are obtained from already calculated DmixScaleSign and DmixScaleTblIndex parameters as follows:

```
InvDmixScale = DmixScaleSign  * InvDmixTable [DmixScaleTblIndex-40];
```

The decoder does not use DmixCoeff and InvDmixScale separately. It uses their product instead. The resulting UndoDmixScale parameter is calculated using fixed-point arithmetic as follows:

```
    UndoDmixScale = ( InvDmixScale * DmixCoeff + (1<<15) ) >>16
```

where << and >> denote left and right integer shift operators. The UndoDmixScale is a 24-bit signed integer represented in Q15 fixed point representation (i.e. all its values are scaled by 2^{15}).

Annex D (normative): Large Tables

D.1 Scale Factor Quantization Tables

D.1.1 6-bit Quantization (Nominal 2,2 dB Step)

Index	Quantization level	Quantization level in dB
0	1	0,0
1	2	6,0
2	2	6,0
3	3	9,5
4	3	9,5
5	4	12,0
6	6	15,5
7	7	17,0
8	10	20,0
9	12	21,5
10	16	24,0
11	20	26,0
12	26	28,3
13	34	30,6
14	44	32,8
15	56	35,0
16	72	37,2
17	93	39,4
18	120	41,6
19	155	43,8
20	200	46,0
21	257	48,2
22	331	50,4
23	427	52,6
24	550	54,8
25	708	57,0
26	912	59,2
27	1175	61,4
28	1514	63,6
29	1950	65,8
30	2512	68,0
31	3236	70,2

Index	Quantization level	Quantization level in dB
32	4169	72,4
33	5370	74,6
34	6918	76,8
35	8913	79,0
36	11482	81,2
37	14791	83,4
38	19055	85,6
39	24547	87,8
40	31623	90,0
41	40738	92,2
42	52481	94,4
43	67608	96,6
44	87096	98,8
45	112202	101,0
46	144544	103,2
47	186209	105,4
48	239883	107,6
49	309030	109,8
50	398107	112,0
51	512861	114,2
52	660693	116,4
53	851138	118,6
54	1096478	120,8
55	1412538	123,0
56	1819701	125,2
57	2344229	127,4
58	3019952	129,6
59	3890451	131,8
60	5011872	134,0
61	6456542	136,2
62	8317638	138,4
63	invalid	invalid

D.1.2 7-bit Quantization (Nominal 1,1 dB Step)

Index	Quantization level	Quantization Level (dB)
0	1	0,0
1	1	0,0
2	2	6,0
3	2	6,0
4	2	6,0
5	2	6,0
6	3	9,5
7	3	9,5
8	3	9,5
9	4	12,0
10	4	12,0
11	5	14,0
12	6	15,5
13	7	17,0
14	7	17,0
15	8	18,0
16	10	20,0
17	11	21,0
18	12	21,5
19	14	23,0
20	16	24,0
21	18	25,1
22	20	26,0
23	23	27,2
24	26	28,3
25	30	29,5
26	34	30,6
27	38	31,6
28	44	32,8
29	50	34,0
30	56	35,0
31	64	36,1
32	72	37,2
33	82	38,3
34	93	39,4
35	106	40,5
36	120	41,6
37	136	42,7
38	155	43,8
39	176	44,9
40	200	46,0
41	226	47,1
42	257	48,2
43	292	49,3
44	331	50,4
45	376	51,5
46	427	52,6
47	484	53,7
48	550	54,8
49	624	55,9
50	708	57,0
51	804	58,1
52	912	59,2
53	1035	60,3
54	1175	61,4
55	1334	62,5
56	1514	63,6
57	1718	64,7
58	1950	65,8
59	2213	66,9
60	2512	68,0
61	2851	69,1
62	3236	70,2
63	3673	71,3

Index	Quantization level	Quantization Level (dB)
64	4169	72,4
65	4732	73,5
66	5370	74,6
67	6095	75,7
68	6918	76,8
69	7852	77,9
70	8913	79,0
71	10116	80,1
72	11482	81,2
73	13032	82,3
74	14791	83,4
75	16788	84,5
76	19055	85,6
77	21627	86,7
78	24547	87,8
79	27861	88,9
80	31623	90,0
81	35892	91,1
82	40738	92,2
83	46238	93,3
84	52481	94,4
85	59566	95,5
86	67608	96,6
87	76736	97,7
88	87096	98,8
89	98855	99,9
90	112202	101,0
91	127350	102,1
92	144544	103,2
93	164059	104,3
94	186209	105,4
95	211349	106,5
96	239883	107,6
97	272270	108,7
98	309030	109,8
99	350752	110,9
100	398107	112,0
101	451856	113,1
102	512861	114,2
103	582103	115,3
104	660693	116,4
105	749894	117,5
106	851138	118,6
107	966051	119,7
108	1096478	120,8
109	1244515	121,9
110	1412538	123,0
111	1603245	124,1
112	1819701	125,2
113	2065380	126,3
114	2344229	127,4
115	2660725	128,5
116	3019952	129,6
117	3427678	130,7
118	3890451	131,8
119	4415704	132,9
120	5011872	134,0
121	5688529	135,1
122	6456542	136,2
123	7328245	137,3
124	8317638	138,4
125	invalid	invalid
126	invalid	invalid
127	invalid	invalid

D.2 Quantization Step Size

D.2.1 Lossy Quantization

ABITS Index	Step-size×2^{22}	Nominal Step-size
0	0	0,0
1	6710886	1,6
2	4194304	1,0
3	3355443	0,8
4	2474639	0,59
5	2097152	0,50
6	1761608	0,42
7	1426063	0,34
8	796918	0,19
9	461373	0,11
10	251658	0,06
11	146801	0,035
12	79692	0,019
13	46137	0,011
14	27263	0,0065
15	16777	0,0040
16	10486	0,0025
17	5872	0,0014
18	3355	0,0008
19	1887	0,00045
20	1258	0,00030
21	713	0,00017
22	336	0,00008
23	168	0,00004
24	84	0,00002
25	42	0,00001
26	21	0,000005
27	invalid	invalid
28	invalid	invalid
29	invalid	invalid
30	invalid	invalid
31	invalid	invalid

D.2.2 Lossless Quantization

ABITS Index	Step-size×2^{22}	Nominal Step-size
0	0	0,0
1	4194304	1,0
2	2097152	0,5
3	1384120	0,33
4	1048576	0,25
5	696254	0,166
6	524288	0,125
7	348127	0,083
8	262144	0,0625
9	131072	0,03125
10	65431	0,0156
11	33026	7,874e-3
12	16450	3,922e-3
13	8208	1,957e-3
14	4100	9,775e-4
15	2049	4,885e-4
16	1024	2,442e-4
17	512	1,221e-4
18	256	6,104e-5
19	128	3,052e-5
20	64	1,526e-5
21	32	7,629e-6
22	16	3,815e-6
23	8	1,907e-6
24	4	9,537e-7
25	2	4,768e-7
26	1	2,384e-7
27	invalid	invalid
28	invalid	invalid
29	invalid	invalid
30	invalid	invalid
31	invalid	invalid

D.3 Scale Factor for Joint Intensity Coding

Index	Scale Factor
0	0,025088
1	0,026624
2	0,02816
3	0,029824
4	0,031616
5	0,033472
6	0,035456
7	0,037568
8	0,039808
9	0,042176
10	0,044672
11	0,047296
12	0,050112
13	0,05312
14	0,056256
15	0,059584
16	0,063104
17	0,066816
18	0,070784
19	0,075008
20	0,079424
21	0,08416
22	0,089152
23	0,0944
24	0,099968
25	0,10592
26	0,112192
27	0,118848
28	0,125888
29	0,133376
30	0,141248
31	0,149632

Index	Scale Factor
32	0,158464
33	0,167872
34	0,177856
35	0,188352
36	0,199552
37	0,211328
38	0,223872
39	0,23712
40	0,2512
41	0,266048
42	0,281856
43	0,29856
44	0,316224
45	0,334976
46	0,354816
47	0,375808
48	0,39808
49	0,421696
50	0,446656
51	0,473152
52	0,501184
53	0,53088
54	0,562368
55	0,595648
56	0,630976
57	0,668352
58	0,707968
59	0,749888
60	0,794304
61	0,841408
62	0,891264
63	0,944064

Index	Scale Factor
64	1
65	1,05926
66	1,12205
67	1,18848
68	1,25894
69	1,3335
70	1,41254
71	1,49626
72	1,5849
73	1,67878
74	1,7783
75	1,88365
76	1,99526
77	2,11347
78	2,23872
79	2,37139
80	2,51187
81	2,66074
82	2,81837
83	2,98541
84	3,1623
85	3,34963
86	3,54816
87	3,7584
88	3,98106
89	4,21696
90	4,46682
91	4,73152
92	5,0119
93	5,30886
94	5,62342
95	5,95661

Index	Scale Factor
96	6,30957
97	6,68346
98	7,07949
99	7,49894
100	7,9433
101	8,41395
102	8,91251
103	9,44064
104	10
105	10,5925
106	11,2202
107	11,885
108	12,5892
109	13,3352
110	14,1254
111	14,9624
112	15,849
113	16,788
114	17,7828
115	18,8365
116	19,9526
117	21,1349
118	22,3872
119	23,7137
120	25,1188
121	26,6072
122	28,1838
123	29,8538
124	31,6228
125	33,4965
126	35,4813
127	37,5837
128	39,8107

D.4 Dynamic Range Control

Index	Q18 binary	Multiplier	Log Multiplier (dB)
0	0,00040394	0,0259	-31,7500
1	0,00041574	0,0266	-31,5000
2	0,00042788	0,0274	-31,2500
3	0,00044037	0,0282	-31,0000
4	0,00045323	0,0290	-30,7500
5	0,00046647	0,0299	-30,5000
6	0,00048009	0,0307	-30,2500
7	0,00049411	0,0316	-30,0000
8	0,00050853	0,0325	-29,7500
9	0,00052338	0,0335	-29,5000
10	0,00053867	0,0345	-29,2500
11	0,00055440	0,0355	-29,0000
12	0,00057058	0,0365	-28,7500
13	0,00058725	0,0376	-28,5000
14	0,00060439	0,0387	-28,2500
15	0,00062204	0,0398	-28,0000
16	0,00064021	0,0410	-27,7500
17	0,00065890	0,0422	-27,5000
18	0,00067814	0,0434	-27,2500
19	0,00069794	0,0447	-27,0000
20	0,00071832	0,0460	-26,7500
21	0,00073930	0,0473	-26,5000
22	0,00076089	0,0487	-26,2500
23	0,00078311	0,0501	-26,0000
24	0,00080597	0,0516	-25,7500
25	0,00082951	0,0531	-25,5000
26	0,00085373	0,0546	-25,2500
27	0,00087866	0,0562	-25,0000
28	0,00090432	0,0579	-24,7500
29	0,00093072	0,0596	-24,5000
30	0,00095790	0,0613	-24,2500
31	0,00098587	0,0631	-24,0000
32	0,00101466	0,0649	-23,7500
33	0,00104429	0,0668	-23,5000
34	0,00107478	0,0688	-23,2500
35	0,00110617	0,0708	-23,0000
36	0,00113847	0,0729	-22,7500
37	0,00117171	0,0750	-22,5000
38	0,00120592	0,0772	-22,2500
39	0,00124114	0,0794	-22,0000
40	0,00127738	0,0818	-21,7500
41	0,00131468	0,0841	-21,5000
42	0,00135307	0,0866	-21,2500
43	0,00139258	0,0891	-21,0000
44	0,00143324	0,0917	-20,7500
45	0,00147510	0,0944	-20,5000
46	0,00151817	0,0972	-20,2500
47	0,00156250	0,1000	-20,0000
48	0,00160813	0,1029	-19,7500
49	0,00165508	0,1059	-19,5000
50	0,00170341	0,1090	-19,2500
51	0,00175315	0,1122	-19,0000
52	0,00180435	0,1155	-18,7500
53	0,00185703	0,1189	-18,5000
54	0,00191126	0,1223	-18,2500
55	0,00196707	0,1259	-18,0000
56	0,00202451	0,1296	-17,7500
57	0,00208363	0,1334	-17,5000
58	0,00214447	0,1372	-17,2500
59	0,00220709	0,1413	-17,0000
60	0,00227154	0,1454	-16,7500
61	0,00233787	0,1496	-16,5000
62	0,00240614	0,1540	-16,2500
63	0,00247640	0,1585	-16,0000

Index	Q18 binary	Multiplier	Log Multiplier (dB)
64	0,00254871	0,1631	-15,7500
65	0,00262313	0,1679	-15,5000
66	0,00269973	0,1728	-15,2500
67	0,00277856	0,1778	-15,0000
68	0,00285970	0,1830	-14,7500
69	0,00294320	0,1884	-14,5000
70	0,00302914	0,1939	-14,2500
71	0,00311760	0,1995	-14,0000
72	0,00320863	0,2054	-13,7500
73	0,00330233	0,2113	-13,5000
74	0,00339876	0,2175	-13,2500
75	0,00349800	0,2239	-13,0000
76	0,00360015	0,2304	-12,7500
77	0,00370527	0,2371	-12,5000
78	0,00381347	0,2441	-12,2500
79	0,00392482	0,2512	-12,0000
80	0,00403943	0,2585	-11,7500
81	0,00415738	0,2661	-11,5000
82	0,00427878	0,2738	-11,2500
83	0,00440372	0,2818	-11,0000
84	0,00453231	0,2901	-10,7500
85	0,00466466	0,2985	-10,5000
86	0,00480087	0,3073	-10,2500
87	0,00494106	0,3162	-10,0000
88	0,00508534	0,3255	-9,7500
89	0,00523383	0,3350	-9,5000
90	0,00538667	0,3447	-9,2500
91	0,00554396	0,3548	-9,0000
92	0,00570585	0,3652	-8,7500
93	0,00587246	0,3758	-8,5000
94	0,00604394	0,3868	-8,2500
95	0,00622042	0,3981	-8,0000
96	0,00640206	0,4097	-7,7500
97	0,00658901	0,4217	-7,5000
98	0,00678141	0,4340	-7,2500
99	0,00697943	0,4467	-7,0000
100	0,00718323	0,4597	-6,7500
101	0,00739299	0,4732	-6,5000
102	0,00760887	0,4870	-6,2500
103	0,00783105	0,5012	-6,0000
104	0,00805972	0,5158	-5,7500
105	0,00829507	0,5309	-5,5000
106	0,00853729	0,5464	-5,2500
107	0,00878658	0,5623	-5,0000
108	0,00904316	0,5788	-4,7500
109	0,00930722	0,5957	-4,5000
110	0,00957900	0,6131	-4,2500
111	0,00985871	0,6310	-4,0000
112	0,01014659	0,6494	-3,7500
113	0,01044287	0,6683	-3,5000
114	0,01074781	0,6879	-3,2500
115	0,01106165	0,7079	-3,0000
116	0,01138466	0,7286	-2,7500
117	0,01171710	0,7499	-2,5000
118	0,01205924	0,7718	-2,2500
119	0,01241138	0,7943	-2,0000
120	0,01277380	0,8175	-1,7500
121	0,01314680	0,8414	-1,5000
122	0,01353069	0,8660	-1,2500
123	0,01392580	0,8913	-1,0000
124	0,01433244	0,9173	-0,7500
125	0,01475095	0,9441	-0,5000
126	0,01518169	0,9716	-0,2500
127	0,01562500	1,0000	0,0000

Index	Q18 binary	Multiplier	Log Multiplier (dB)
128	0,01608126	1,0292	0,2500
129	0,01655084	1,0593	0,5000
130	0,01703413	1,0902	0,7500
131	0,01753154	1,1220	1,0000
132	0,01804347	1,1548	1,2500
133	0,01857035	1,1885	1,5000
134	0,01911261	1,2232	1,7500
135	0,01967071	1,2589	2,0000
136	0,02024510	1,2957	2,2500
137	0,02083627	1,3335	2,5000
138	0,02144470	1,3725	2,7500
139	0,02207090	1,4125	3,0000
140	0,02271538	1,4538	3,2500
141	0,02337868	1,4962	3,5000
142	0,02406135	1,5399	3,7500
143	0,02476396	1,5849	4,0000
144	0,02548708	1,6312	4,2500
145	0,02623131	1,6788	4,5000
146	0,02699728	1,7278	4,7500
147	0,02778562	1,7783	5,0000
148	0,02859697	1,8302	5,2500
149	0,02943202	1,8836	5,5000
150	0,03029145	1,9387	5,7500
151	0,03117597	1,9953	6,0000
152	0,03208633	2,0535	6,2500
153	0,03302327	2,1135	6,5000
154	0,03398756	2,1752	6,7500
155	0,03498002	2,2387	7,0000
156	0,03600145	2,3041	7,2500
157	0,03705271	2,3714	7,5000
158	0,03813467	2,4406	7,7500
159	0,03924823	2,5119	8,0000
160	0,04039429	2,5852	8,2500
161	0,04157383	2,6607	8,5000
162	0,04278781	2,7384	8,7500
163	0,04403723	2,8184	9,0000
164	0,04532314	2,9007	9,2500
165	0,04664660	2,9854	9,5000
166	0,04800871	3,0726	9,7500
167	0,04941059	3,1623	10,0000
168	0,05085340	3,2546	10,2500
169	0,05233835	3,3497	10,5000
170	0,05386666	3,4475	10,7500
171	0,05543959	3,5481	11,0000
172	0,05705846	3,6517	11,2500
173	0,05872459	3,7584	11,5000
174	0,06043938	3,8681	11,7500
175	0,06220425	3,9811	12,0000
176	0,06402064	4,0973	12,2500
177	0,06589008	4,2170	12,5000
178	0,06781410	4,3401	12,7500
179	0,06979431	4,4668	13,0000
180	0,07183234	4,5973	13,2500
181	0,07392988	4,7315	13,5000
182	0,07608868	4,8697	13,7500
183	0,07831051	5,0119	14,0000
184	0,08059721	5,1582	14,2500
185	0,08295069	5,3088	14,5000
186	0,08537290	5,4639	14,7500
187	0,08786583	5,6234	15,0000
188	0,09043156	5,7876	15,2500
189	0,09307221	5,9566	15,5000
190	0,09578997	6,1306	15,7500
191	0,09858709	6,3096	16,0000

Index	Q18 binary	Multiplier	Log Multiplier (dB)
192	0,10146588	6,4938	16,2500
193	0,10442874	6,6834	16,5000
194	0,10747811	6,8786	16,7500
195	0,11061653	7,0795	17,0000
196	0,11384659	7,2862	17,2500
197	0,11717097	7,4989	17,5000
198	0,12059242	7,7179	17,7500
199	0,12411379	7,9433	18,0000
200	0,12773797	8,1752	18,2500
201	0,13146799	8,4140	18,5000
202	0,13530693	8,6596	18,7500
203	0,13925796	8,9125	19,0000
204	0,14332436	9,1728	19,2500
205	0,14750951	9,4406	19,5000
206	0,15181687	9,7163	19,7500
207	0,15625000	10,0000	20,0000
208	0,16081258	10,2920	20,2500
209	0,16550839	10,5925	20,5000
210	0,17034133	10,9018	20,7500
211	0,17531538	11,2202	21,0000
212	0,18043469	11,5478	21,2500
213	0,18570347	11,8850	21,5000
214	0,19112611	12,2321	21,7500
215	0,19670710	12,5893	22,0000
216	0,20245105	12,9569	22,2500
217	0,20836272	13,3352	22,5000
218	0,21444703	13,7246	22,7500
219	0,22070899	14,1254	23,0000
220	0,22715381	14,5378	23,2500
221	0,23378682	14,9624	23,5000
222	0,24061352	15,3993	23,7500
223	0,24763956	15,8489	24,0000
224	0,25487077	16,3117	24,2500
225	0,26231313	16,7880	24,5000
226	0,26997281	17,2783	24,7500
227	0,27785616	17,7828	25,0000
228	0,28596970	18,3021	25,2500
229	0,29432017	18,8365	25,5000
230	0,30291447	19,3865	25,7500
231	0,31175974	19,9526	26,0000
232	0,32086329	20,5353	26,2500
233	0,33023266	21,1349	26,5000
234	0,33987563	21,7520	26,7500
235	0,34980018	22,3872	27,0000
236	0,36001453	23,0409	27,2500
237	0,37052714	23,7137	27,5000
238	0,38134673	24,4062	27,7500
239	0,39248225	25,1189	28,0000
240	0,40394294	25,8523	28,2500
241	0,41573829	26,6073	28,5000
242	0,42787807	27,3842	28,7500
243	0,44037233	28,1838	29,0000
244	0,45323144	29,0068	29,2500
245	0,46646603	29,8538	29,5000
246	0,48008709	30,7256	29,7500
247	0,49410588	31,6228	30,0000
248	0,50853404	32,5462	30,2500
249	0,52338350	33,4965	30,5000
250	0,53866657	34,4747	30,7500
251	0,55439592	35,4813	31,0000
252	0,57058457	36,5174	31,2500
253	0,58724594	37,5837	31,5000
254	0,60439384	38,6812	31,7500
255	0,62204245	39,8107	32,0000

D.5 Huffman Code Books

D.5.1 3 Levels

Table A3

Quantization level	Code length	Code
0	1	0
1	2	2
-1	2	3

D.5.2 4 Levels (For TMODE)

Table A4

Quantization level	Code length	Code
0	1	0
1	2	2
2	3	6
3	3	7

Table B4

Quantization level	Code length	Code
0	2	2
1	3	6
2	3	7
3	1	0

Table C4

Quantization level	Code length	Code
0	3	6
1	3	7
2	1	0
3	2	2

Table D4

Quantization level	Code length	Code
0	2	0
1	2	1
2	2	2
3	2	3

D.5.3 5 Levels

Table A5

Quantization level	Code length	Code
0	1	0
1	2	2
-1	3	6
2	4	14
-2	4	15

Table B5

Quantization level	Code length	Code
0	2	2
1	2	0
-1	2	1
2	3	6
-2	3	7

Table C5

Quantization level	Code length	Code
0	1	0
1	3	4
-1	3	5
2	3	6
-2	3	7

D.5.4 7 Levels

Table A7

Quantization level	Code length	Code
0	1	0
1	3	6
-1	3	5
2	3	4
-2	4	14
3	5	31
-3	5	30

Table B7

Quantization level	Code length	Code
0	2	3
1	2	1
-1	2	0
2	3	4
-2	4	11
3	5	21
-3	5	20

Table C7

Quantization level	Code length	Code
0	2	3
1	2	2
-1	2	1
2	4	3
-2	4	2
3	4	1
-3	4	0

D.5.5 9 Levels

Table A9

Quantization level	Code length	Code
0	1	0
1	3	7
-1	3	5
2	4	13
-2	4	9
3	4	8
-3	5	25
4	6	49
-4	6	48

Table B9

Quantization level	Code length	Code
0	2	2
1	2	0
-1	3	7
2	3	3
-2	3	2
3	5	27
-3	5	26
4	5	25
-4	5	24

Table C9

Quantization level	Code length	Code
0	2	2
1	2	0
-1	3	7
2	3	6
-2	3	2
3	4	6
-3	5	15
4	6	29
-4	6	28

D.5.6 12 Levels (for BHUFF)

Table A12

ABITS	Code length	Code
1	1	0
2	2	2
3	3	6
4	4	14
5	5	30
6	6	62
7	8	255
8	8	254
9	9	507
10	9	506
11	9	505
12	9	504

Table B12

ABITS	Code length	Code
1	1	1
2	2	0
3	3	2
4	5	15
5	5	12
6	6	29
7	7	57
8	7	56
9	7	55
10	7	54
11	7	53
12	7	52

Table C12

ABITS	Code length	Code
1	2	0
2	3	7
3	3	5
4	3	4
5	3	2
6	4	13
7	4	12
8	4	6
9	5	15
10	6	29
11	7	57
12	7	56

Table D12

ABITS	Code length	Code
1	2	3
2	2	2
3	2	0
4	3	2
5	4	6
6	5	14
7	6	30
8	7	62
9	8	126
10	9	254
11	10	511
12	10	510

Table E12

ABITS	Code length	Code
1	1	1
2	2	0
3	3	2
4	4	6
5	5	14
6	7	63
7	7	61
8	8	124
9	8	121
10	8	120
11	9	251
12	9	250

D.5.7 13 Levels

Table A13

Quantization level	Code length	Code
0	1	0
1	3	4
-1	4	15
2	4	13
-2	4	12
3	4	10
-3	5	29
4	5	22
-4	6	57
5	6	47
-5	6	46
6	7	113
-6	7	112

Table B13

Quantization level	Code length	Code
0	2	0
1	3	6
-1	3	5
2	3	2
-2	4	15
3	4	9
-3	4	7
4	4	6
-4	5	29
5	5	17
-5	5	16
6	6	57
-6	6	56

Table C13

Quantization level	Code length	Code
0	3	5
1	3	4
-1	3	3
2	3	2
-2	3	0
3	4	15
-3	4	14
4	4	12
-4	4	3
5	5	27
-5	5	26
6	5	5
-6	5	4

D.5.8 17 Levels

Table A17

Quantization level	Code length	Code
0	2	1
1	3	7
-1	3	6
2	3	4
-2	3	1
3	4	11
-3	4	10
4	4	0
-4	5	3
5	6	4
-5	7	11
6	8	20
-6	9	43
7	10	84
-7	11	171
8	12	341
-8	12	340

Table B17

Quantization level	Code length	Code
0	2	0
1	3	6
-1	3	5
2	3	2
-2	4	15
3	4	9
-3	4	8
4	5	29
-4	5	28
5	5	14
-5	5	13
6	6	30
-6	6	25
7	6	24
-7	7	63
8	8	125
-8	8	124

Table C17

Quantization level	Code length	Code
0	3	6
1	3	4
-1	3	3
2	3	0
-2	4	15
3	4	11
-3	4	10
4	4	4
-4	4	3
5	5	29
-5	5	28
6	5	10
-6	5	5
7	5	4
-7	6	23
8	7	45
-8	7	44

Table D17

Quantization level	Code length	Code
0	1	0
1	3	7
-1	3	6
2	4	11
-2	4	10
3	5	19
-3	5	18
4	6	35
-4	6	34
5	7	67
-5	7	66
6	8	131
-6	8	130
7	9	259
-7	9	258
8	9	257
-8	9	256

Table E17

Quantization level	Code length	Code
0	1	0
1	3	5
-1	3	4
2	4	12
-2	5	31
3	5	28
-3	5	27
4	6	60
-4	6	59
5	6	53
-5	6	52
6	7	122
-6	7	117
7	8	247
-7	8	246
8	8	233
-8	8	232

Table F17

Quantization level	Code length	Code
0	3	6
1	3	5
-1	3	4
2	3	2
-2	3	1
3	4	15
-3	4	14
4	4	6
-4	4	1
5	5	14
-5	5	1
6	6	31
-6	6	30
7	6	0
-7	7	3
8	8	5
-8	8	4

Table G17

Quantization level	Code length	Code
0	2	2
1	3	7
-1	3	6
2	3	1
-2	3	0
3	4	5
-3	4	4
4	5	14
-4	5	13
5	6	30
-5	6	25
6	7	62
-6	7	49
7	8	127
-7	8	126
8	8	97
-8	8	96

D.5.9 25 Levels

Table A25

Quantization level	Code length	Code
0	3	6
1	3	4
-1	3	3
2	3	1
-2	3	0
3	4	15
-3	4	14
4	4	5
-4	4	4
5	5	22
-5	5	21
6	6	47
-6	6	46
7	7	83
-7	7	82
8	8	163
-8	8	162
9	8	160
-9	9	323
10	10	644
-10	11	1 291
11	12	2 580
-11	13	5 163
12	14	10 325
-12	14	10 324

Table B25

Quantization level	Code length	Code
0	3	5
1	3	2
-1	3	1
2	4	15
-2	4	14
3	4	9
-3	4	8
4	4	6
-4	4	1
5	5	26
-5	5	25
6	5	15
-6	5	14
7	6	55
-7	6	54
8	6	49
-8	6	48
9	6	1
-9	6	0
10	7	6
-10	7	5
11	7	4
-11	8	15
12	9	29
-12	9	28

Table C25

Quantization level	Code length	Code
0	3	1
1	4	15
-1	4	14
2	4	12
-2	4	11
3	4	9
-3	4	8
4	4	6
-4	4	5
5	4	1
-5	4	0
6	5	26
-6	5	21
7	5	15
-7	5	14
8	5	8
-8	6	55
9	6	41
-9	6	40
10	6	18
-10	7	109
11	7	108
-11	7	39
12	8	77
-12	8	76

Table D25

Quantization level	Code length	Code
0	2	2
1	3	7
-1	3	6
2	3	1
-2	3	0
3	4	5
-3	4	4
4	5	13
-4	5	12
5	6	29
-5	6	28
6	7	62
-6	7	61
7	8	126
-7	8	121
8	9	255
-8	9	254
9	10	483
-9	10	482
10	11	963
-10	11	962
11	12	1 923
-11	12	1 922
12	12	1 921
-12	12	1 920

Table E25

Quantization level	Code length	Code
0	2	3
1	3	3
-1	3	2
2	4	11
-2	4	10
3	4	1
-3	4	0
4	5	17
-4	5	16
5	5	5
-5	5	4
6	6	38
-6	6	37
7	6	14
-7	6	13
8	7	79
-8	7	78
9	7	72
-9	7	31
10	7	25
-10	7	24
11	8	147
-11	8	146
12	8	61
-12	8	60

Table F25

Quantization level	Code length	Code
0	3	1
1	3	0
-1	4	15
2	4	14
-2	4	13
3	4	11
-3	4	10
4	4	8
-4	4	7
5	4	5
-5	4	4
6	5	24
-6	5	19
7	5	13
-7	5	12
8	6	37
-8	6	36
9	7	102
-9	7	101
10	8	207
-10	8	206
11	8	200
-11	9	403
12	10	805
-12	10	804

Table G25

Quantization level	Code length	Code
0	2	1
1	3	6
-1	3	5
2	3	0
-2	4	15
3	4	8
-3	4	3
4	5	28
-4	5	19
5	5	4
-5	6	59
6	6	36
-6	6	11
7	7	116
-7	7	75
8	7	21
-8	7	20
9	8	149
-9	8	148
10	9	470
-10	9	469
11	10	943
-11	10	942
12	10	937
-12	10	936

D.5.10 33 Levels

Table A33

Quantization level	Code length	Code
0	3	2
1	3	1
-1	3	0
2	4	14
-2	4	13
3	4	12
-3	4	11
4	4	9
-4	4	8
5	4	6
-5	5	31
6	5	20
-6	5	15
7	6	61
-7	6	60
8	6	29
-8	6	28
9	7	85
-9	7	84
10	8	174
-10	8	173
11	9	351
-11	9	350
12	10	691
-12	10	690
13	11	1 379
-13	11	1 378
14	12	2 755
-14	12	2 754
15	13	5 507
-15	13	5 506
16	13	5 505
-16	13	5 504

Table B33

Quantization level	Code length	Code
0	3	1
1	4	15
-1	4	14
2	4	11
-2	4	10
3	4	8
-3	4	7
4	4	4
-4	4	1
5	5	27
-5	5	26
6	5	19
-6	5	18
7	5	12
-7	5	11
8	5	1
-8	5	0
9	6	50
-9	6	49
10	6	26
-10	6	21
11	7	103
-11	7	102
12	7	96
-12	7	55
13	7	41
-13	7	40
14	8	194
-14	8	109
15	8	108
-15	9	391
16	10	781
-16	10	780

Table C33

Quantization Level	Code length	Code
0	4	13
1	4	11
-1	4	10
2	4	8
-2	4	7
3	4	4
-3	4	3
4	4	2
-4	4	1
5	5	30
-5	5	29
6	5	25
-6	5	24
7	5	19
-7	5	18
8	5	11
-8	5	10
9	5	0
-9	6	63
10	6	62
-10	6	57
11	6	27
-11	6	26
12	6	24
-12	6	3
13	7	113
-13	7	112
14	7	50
-14	7	5
15	7	4
-15	8	103
16	9	205
-16	9	204

Table D33

Quantization level	Code length	Code
0	2	1
1	3	6
-1	3	5
2	3	0
-2	4	15
3	4	8
-3	4	3
4	5	28
-4	5	19
5	5	4
-5	6	59
6	6	36
-6	6	11
7	7	116
-7	7	75
8	7	21
-8	7	20
9	8	149
-9	8	148
10	9	469
-10	9	468
11	10	941
-11	10	940
12	11	1 885
-12	11	1 884
13	12	3 773
-13	12	3 772
14	13	7 551
-14	13	7 550
15	14	15 099
-15	14	15 098
16	14	15 097
-16	14	15 096

Table E33

Quantization level	Code length	Code
0	2	2
1	3	2
-1	3	1
2	4	12
-2	4	7
3	4	0
-3	5	31
4	5	27
-4	5	26
5	5	3
-5	5	2
6	6	59
-6	6	58
7	6	27
-7	6	26
8	7	123
-8	7	122
9	7	120
-9	7	115
10	7	112
-10	7	51
11	7	49
-11	7	48
12	8	242
-12	8	229
13	8	227
-13	8	226
14	8	101
-14	8	100
15	9	487
-15	9	486
16	9	457
-16	9	456

Table F33

Quantization level	Code length	Code
0	4	13
1	4	12
-1	4	11
2	4	9
-2	4	8
3	4	7
-3	4	6
4	4	4
-4	4	3
5	4	1
-5	4	0
6	5	30
-6	5	29
7	5	21
-7	5	20
8	5	10
-8	5	5
9	6	63
-9	6	62
10	6	56
-10	6	23
11	6	9
-11	6	8
12	7	45
-12	7	44
13	8	230
-13	8	229
14	9	463
-14	9	462
15	9	456
-15	10	915
16	11	1 829
-16	11	1 828

Table G33

Quantization level	Code length	Code
0	3	6
1	3	3
-1	3	2
2	4	15
-2	4	14
3	4	9
-3	4	8
4	4	1
-4	4	0
5	5	22
-5	5	21
6	5	6
-6	5	5
7	6	46
-7	6	41
8	6	14
-8	6	9
9	7	94
-9	7	81
10	7	30
-10	7	17
11	8	191
-11	8	190
12	8	63
-12	8	62
13	8	32
-13	9	323
14	9	321
-14	9	320
15	9	67
-15	9	66
16	10	645
-16	10	644

D.5.11 65 Levels

Table A65

Quantization level	Code length	Code
0	4	6
1	4	5
-1	4	4
2	4	2
-2	4	1
3	4	0
-3	5	31
4	5	29
-4	5	28
5	5	27
-5	5	26
6	5	24
-6	5	23
7	5	21
-7	5	20
8	5	18
-8	5	17
9	5	14
-9	5	7
10	5	6
-10	6	61
11	6	50
-11	6	45
12	6	38
-12	6	33
13	6	31
-13	6	30
14	7	120
-14	7	103
15	7	89
-15	7	88
16	7	65
-16	7	64
17	8	205
-17	8	204
18	8	157
-18	8	156
19	9	486
-19	9	485
20	9	318
-20	9	317
21	10	975
-21	10	974
22	10	639
-22	10	638
23	11	1 939
-23	11	1 938
24	11	1 936
-24	11	1 267
25	11	1 264
-25	12	3 875
26	12	2 532
-26	12	2 531
27	13	7 749
-27	13	7 748
28	13	5 061
-28	13	5 060
29	14	10 133
-29	14	10 132

Quantization level	Code length	Code
30	15	20 269
-30	15	20 268
31	16	40 543
-31	16	40 542
32	16	40 541
-32	16	40 540

Table B65

Quantization level	Code length	Code
0	4	4
1	4	2
-1	4	1
2	5	30
-2	5	29
3	5	26
-3	5	25
4	5	23
-4	5	22
5	5	19
-5	5	18
6	5	16
-6	5	15
7	5	12
-7	5	11
8	5	7
-8	5	6
9	6	63
-9	6	62
10	6	56
-10	6	55
11	6	49
-11	6	48
12	6	41
-12	6	40
13	6	34
-13	6	29
14	6	26
-14	6	21
15	6	20
-15	6	3
16	6	0
-16	7	115
17	7	109
-17	7	108
18	7	86
-18	7	85
19	7	70
-19	7	57
20	7	56
-20	7	55
21	7	4
-21	7	3
22	8	229
-22	8	228
23	8	175
-23	8	174
24	8	143
-24	8	142
25	8	108
-25	8	11
26	8	10
-26	8	5

Quantization level	Code length	Code
27	9	339
-27	9	338
28	9	336
-28	9	219
29	9	9
-29	9	8
30	10	674
-30	10	437
31	10	436
-31	11	1 351
32	12	2 701
-32	12	2 700

Table C65

Quantization level	Code length	Code
0	5	28
1	5	25
-1	5	24
2	5	23
-2	5	22
3	5	19
-3	5	18
4	5	16
-4	5	15
5	5	13
-5	5	12
6	5	10
-6	5	9
7	5	7
-7	5	6
8	5	4
-8	5	3
9	5	1
-9	5	0
10	6	62
-10	6	61
11	6	59
-11	6	58
12	6	54
-12	6	53
13	6	43
-13	6	42
14	6	40
-14	6	35
15	6	29
-15	6	28
16	6	17
-16	6	16
17	6	11
-17	6	10
18	6	4
-18	7	127
19	7	121
-19	7	120
20	7	110
-20	7	105
21	7	83
-21	7	82
22	7	68
-22	7	47
23	7	46
-23	7	45

Quantization level	Code length	Code
24	7	11
-24	7	10
25	8	252
-25	8	223
26	8	209
-26	8	208
27	8	138
-27	8	89
28	8	88
-28	9	507
29	9	445
-29	9	444
30	9	278
-30	10	1013
31	10	1012
-31	10	559
32	11	1117
-32	11	1116

Table D65

Quantization level	Code length	Code
0	3	4
1	3	1
-1	3	0
2	4	13
-2	4	12
3	4	7
-3	4	6
4	5	31
-4	5	30
5	5	23
-5	5	22
6	5	11
-6	5	10
7	6	59
-7	6	58
8	6	43
-8	6	42
9	6	19
-9	6	18
10	7	115
-10	7	114
11	7	83
-11	7	82
12	7	35
-12	7	34
13	8	227
-13	8	226
14	8	163
-14	8	162
15	8	160
-15	8	67
16	8	64
-16	9	451
17	9	448
-17	9	323
18	9	132
-18	9	131
19	10	900
-19	10	899
20	10	644
-20	10	267

Quantization level	Code length	Code
21	10	261
-21	10	260
22	11	1 797
-22	11	1 796
23	11	533
-23	11	532
24	12	3 605
-24	12	3 604
25	12	2 582
-25	12	2 581
26	13	7 215
-26	13	7 214
27	13	5 167
-27	13	5 166
28	13	5 160
-28	14	14 427
29	14	10 323
-29	14	10 322
30	15	28 853
-30	15	28 852

Table E65

Quantization level	Code length	Code
0	3	4
1	3	0
-1	4	15
2	4	7
-2	4	6
3	5	29
-3	5	28
4	5	23
-4	5	22
5	5	10
-5	5	9
6	5	6
-6	5	5
7	6	54
-7	6	53
8	6	48
-8	6	43
9	6	40
-9	6	23
10	6	16
-10	6	15
11	6	9
-11	6	8
12	7	105
-12	7	104
13	7	100
-13	7	99
14	7	84
-14	7	83
15	7	45
-15	7	44
16	7	29
-16	7	28
17	8	221
-17	8	220
18	8	206
-18	8	205
19	8	202
-19	8	197

Quantization level	Code length	Code
20	8	171
-20	8	170
21	8	164
-21	8	71
22	8	69
-22	8	68
23	9	446
-23	9	445
24	9	415
-24	9	414
25	9	408
-25	9	407
26	9	393
-26	9	392
27	9	331
-27	9	330
28	9	141
-28	9	140
29	10	895
-29	10	894
30	10	889
-30	10	888
31	10	819
-31	10	818
32	10	813
-32	10	812

Table F65

Quantization level	Code length	Code
0	3	6
1	3	3
-1	3	2
2	4	15
-2	4	14
3	4	9
-3	4	8
4	4	1
-4	4	0
5	5	21
-5	5	20
6	5	5
-6	5	4
7	6	45
-7	6	44
8	6	13
-8	6	12
9	7	93
-9	7	92
10	7	29
-10	7	28
11	8	189
-11	8	188
12	8	61
-12	8	60
13	9	381
-13	9	380
14	9	125
-14	9	124
15	10	765
-15	10	764
16	10	252
-16	11	1 535

Quantization level	Code length	Code
17	11	1 532
-17	11	511
18	11	506
-18	12	3 069
19	12	3 067
-19	12	3 066
20	12	1 015
-20	12	1 014
21	13	6 136
-21	13	2 043
22	13	2 035
-22	13	2 034
23	14	12 275
-23	14	12 274
24	14	4 085
-24	14	4 084
25	14	4 083
-25	14	4 082
26	14	4 081
-26	14	4 080
27	14	4 079
-27	14	4 078
28	14	4 077
-28	14	4 076
29	14	4 075
-29	14	4 074
30	14	4 073
-30	14	4 072
31	14	4 067
-31	14	4 066
32	14	4 065
-32	14	4 064

Table G65

Quantization level	Code length	Code
0	4	14
1	4	11
-1	4	10
2	4	8
-2	4	6
3	4	4
-3	4	3
4	4	0
-4	5	31
5	5	26
-5	5	25
6	5	18
-6	5	15
7	5	10
-7	5	5
8	5	2
-8	6	61
9	6	54
-9	6	49
10	6	38
-10	6	29
11	6	22
-11	6	9
12	6	6
-12	7	121
13	7	110
-13	7	97

Quantization level	Code length	Code
14	7	78
-14	7	57
15	7	46
-15	7	17
16	7	14
-16	8	241
17	8	223
-17	8	222
18	8	159
-18	8	158
19	8	95
-19	8	94
20	8	31
-20	8	30
21	9	480
-21	9	387
22	9	384
-22	9	227
23	9	225
-23	9	224
24	9	65
-24	9	64
25	10	962
-25	10	773
26	10	771
-26	10	770
27	10	452
-27	10	135
28	10	133
-28	10	132
29	11	1 927
-29	11	1 926
30	11	1 545
-30	11	1 544
31	11	907
-31	11	906
32	11	269
-32	11	268

D.5.12 129 Levels

Table SA129

Quantization level	Code length	Code
0	2	1
1	3	6
-1	3	5
2	3	0
-2	4	15
3	4	8
-3	4	3
4	5	28
-4	5	19
5	5	4
-5	6	59
6	6	36
-6	6	11
7	7	75
-7	7	74
8	8	233
-8	8	232
9	8	41
-9	8	40
10	9	87
-10	9	86
11	10	937
-11	10	936
12	11	1 877
-12	11	1 876
13	11	341
-13	11	340
14	12	686
-14	12	685
15	13	1 375
-15	13	1 374
16	13	1 369
-16	13	1 368
17	13	1 359
-17	13	1 358
18	13	1 357
-18	13	1 356
19	13	1 355
-19	13	1 354
20	13	1 353
-20	13	1 352
21	13	1 351
-21	13	1 350
22	13	1 349
-22	13	1 348
23	13	1 347
-23	13	1 346
24	13	1 345
-24	13	1 344
25	14	15 103
-25	14	15 102
26	14	15 101
-26	14	15 100
27	14	15 099
-27	14	15 098
28	14	15 097
-28	14	15 096
29	14	15 095
-29	14	15 094
30	14	15 093
-30	14	15 092
31	14	15 091
-31	14	15 090
32	14	15 089
-32	14	15 088
33	14	15 087
-33	14	15 086
34	14	15 085
-34	14	15 084
35	14	15 083
-35	14	15 082
36	14	15 081
-36	14	15 080
37	14	15 079
-37	14	15 078
38	14	15 077
-38	14	15 076
39	14	15 075
-39	14	15 074
40	14	15 073
-40	14	15 072
41	14	15 071
-41	14	15 070
42	14	15 069
-42	14	15 068
43	14	15 067
-43	14	15 066
44	14	15 065
-44	14	15 064
45	14	15 063
-45	14	15 062
46	14	15 061
-46	14	15 060
47	14	15 059
-47	14	15 058
48	14	15 057
-48	14	15 056
49	14	15 055
-49	14	15 054
50	14	15 053
-50	14	15 052
51	14	15 051
-51	14	15 050
52	14	15 049
-52	14	15 048
53	14	15 047
-53	14	15 046
54	14	15 045
-54	14	15 044
55	14	15 043
-55	14	15 042
56	14	15 041
-56	14	15 040
57	14	15 039
-57	14	15 038
58	14	15 037
-58	14	15 036
59	14	15 035
-59	14	15 034
60	14	15 033
-60	14	15 032
61	14	15 031
-61	14	15 030
62	14	15 029
-62	14	15 028
63	14	15 027
-63	14	15 026
64	14	15 025
-64	14	15 024

Table SB129

Quantization level	Code length	Code
0	3	3
1	3	2
-1	3	1
2	4	15
-2	4	14
3	4	12
-3	4	11
4	4	10
-4	4	9
5	4	0
-5	5	27
6	5	17
-6	5	16
7	6	53
-7	6	52
8	6	5
-8	6	4
9	7	13
-9	7	12
10	8	29
-10	8	28
11	9	60
-11	10	127
12	11	253
-12	11	252
13	12	491
-13	12	490
14	13	979
-14	13	978
15	14	1 955
-15	14	1 954
16	14	1 953
-16	14	1 952
17	15	4 031
-17	15	4 030
18	15	4 029
-18	15	4 028
19	15	4 027
-19	15	4 026
20	15	4 025
-20	15	4 024
21	15	4 023
-21	15	4 022
22	15	4 021
-22	15	4 020
23	15	4 019
-23	15	4 018
24	15	4 017
-24	15	4 016
25	15	4 015
-25	15	4 014
26	15	4 013
-26	15	4 012
27	15	4 011
-27	15	4 010
28	15	4 009
-28	15	4 008
29	15	4 007
-29	15	4 006
30	15	4 005
-30	15	4 004
31	15	4 003
-31	15	4 002
32	15	4 001
-32	15	4 000
33	15	3 999
-33	15	3 998
34	15	3 997
-34	15	3 996
35	15	3 995
-35	15	3 994
36	15	3 993
-36	15	3 992
37	15	3 991
-37	15	3 990
38	15	3 989
-38	15	3 988
39	15	3 987
-39	15	3 986
40	15	3 985
-40	15	3 984
41	15	3 983
-41	15	3 982
42	15	3 981
-42	15	3 980
43	15	3 979
-43	15	3 978
44	15	3 977
-44	15	3 976
45	15	3 975
-45	15	3 974
46	15	3 973
-46	15	3 972
47	15	3 971
-47	15	3 970
48	15	3 969
-48	15	3 968
49	15	3 967
-49	15	3 966
50	15	3 965
-50	15	3 964
51	15	3 963
-51	15	3 962
52	15	3 961
-52	15	3 960
53	15	3 959
-53	15	3 958
54	15	3 957
-54	15	3 956
55	15	3 955
-55	15	3 954
56	15	3 953
-56	15	3 952
57	15	3 951
-57	15	3 950
58	15	3 949
-58	15	3 948
59	15	3 947
-59	15	3 946
60	15	3 945
-60	15	3 944
61	15	3 943
-61	15	3 942
62	15	3 941
-62	15	3 940
63	15	3 939
-63	15	3 938
64	15	3 937
-64	15	3 936

Table SC129

Quantization level	Code length	Code
0	3	4
1	3	1
-1	3	0
2	4	13
-2	4	12
3	4	7
-3	4	6
4	5	31
-4	5	30
5	5	23
-5	5	22
6	5	11
-6	5	10
7	6	59
-7	6	58
8	6	43
-8	6	42
9	6	19
-9	6	18
10	7	115
-10	7	114
11	7	83
-11	7	82
12	7	35
-12	7	34
13	8	227
-13	8	226
14	8	162
-14	8	161
15	8	66
-15	8	65
16	9	450
-16	9	449
17	9	321
-17	9	320
18	9	129
-18	9	128
19	10	897
-19	10	896
20	10	652
-20	10	271
21	10	268
-21	11	1 807
22	11	1 308
-22	11	1 307
23	11	540
-23	11	539
24	12	3 612
-24	12	3 611
25	12	2 613
-25	12	2 612
26	12	1 077
-26	12	1 076
27	13	7 226
-27	13	7 221
28	13	2 167
-28	13	2 166
29	13	2 164
-29	14	14 455
30	14	14 441
-30	14	14 440
31	14	4 331
-31	14	4 330
32	15	28 909
-32	15	28 908
33	15	28 879
-33	15	28 878
34	15	28 877
-34	15	28 876
35	15	28 875
-35	15	28 874
36	15	28 873
-36	15	28 872
37	15	28 871
-37	15	28 870
38	15	28 869
-38	15	28 868
39	15	28 867
-39	15	28 866
40	15	28 865
-40	15	28 864
41	15	20 991
-41	15	20 990
42	15	20 989
-42	15	20 988
43	15	20 987
-43	15	20 986
44	15	20 985
-44	15	20 984
45	15	20 983
-45	15	20 982
46	15	20 981
-46	15	20 980
47	15	20 979
-47	15	20 978
48	15	20 977
-48	15	20 976
49	15	20 975
-49	15	20 974
50	15	20 973
-50	15	20 972
51	15	20 971
-51	15	20 970
52	15	20 969
-52	15	20 968
53	15	20 967
-53	15	20 966
54	15	20 965
-54	15	20 964
55	15	20 963
-55	15	20 962
56	15	20 961
-56	15	20 960
57	15	20 959
-57	15	20 958
58	15	20 957
-58	15	20 956
59	15	20 955
-59	15	20 954
60	15	20 953
-60	15	20 952
61	15	20 951
-61	15	20 950
62	15	20 949
-62	15	20 948
63	15	20 947
-63	15	20 946
64	15	20 945
-64	15	20 944

Table SD129

Quantization level	Code length	Code
0	2	0
1	3	5
-1	3	4
2	4	15
-2	4	14
3	4	7
-3	4	6
4	5	26
-4	5	25
5	5	10
-5	5	9
6	6	54
-6	6	49
7	6	22
-7	6	17
8	7	110
-8	7	97
9	7	46
-9	7	33
10	8	193
-10	8	192
11	8	65
-11	8	64
12	9	444
-12	9	191
13	9	188
-13	10	895
14	10	890
-14	10	381
15	10	378
-15	11	1 789
16	11	761
-16	11	760
17	12	3 577
-17	12	3 576
18	12	1 519
-18	12	1 518
19	12	1 516
-19	13	7 151
20	13	7 128
-20	13	3 035
21	14	14 301
-21	14	14 300
22	14	6 069
-22	14	6 068
23	15	28 599
-23	15	28 598
24	15	28 597
-24	15	28 596
25	15	28 595
-25	15	28 594
26	15	28 593
-26	15	28 592
27	15	28 591
-27	15	28 590
28	15	28 589
-28	15	28 588
29	15	28 587
-29	15	28 586
30	15	28 585
-30	15	28 584
31	15	28 583
-31	15	28 582
32	15	28 581
-32	15	28 580
33	15	28 579
-33	15	28 578
34	15	28 577
-34	15	28 576
35	15	28 575
-35	15	28 574
36	15	28 573
-36	15	28 572
37	15	28 571
-37	15	28 570
38	15	28 569
-38	15	28 568
39	15	28 567
-39	15	28 566
40	15	28 565
-40	15	28 564
41	15	28 563
-41	15	28 562
42	15	28 561
-42	15	28 560
43	15	28 559
-43	15	28 558
44	15	28 557
-44	15	28 556
45	15	28 555
-45	15	28 554
46	15	28 553
-46	15	28 552
47	15	28 551
-47	15	28 550
48	15	28 549
-48	15	28 548
49	15	28 547
-49	15	28 546
50	15	28 545
-50	15	28 544
51	15	28 543
-51	15	28 542
52	15	28 541
-52	15	28 540
53	15	28 539
-53	15	28 538
54	15	28 537
-54	15	28 536
55	15	28 535
-55	15	28 534
56	15	28 533
-56	15	28 532
57	15	28 531
-57	15	28 530
58	15	28 529
-58	15	28 528
59	15	28 527
-59	15	28 526
60	15	28 525
-60	15	28 524
61	15	28 523
-61	15	28 522
62	15	28 521
-62	15	28 520
63	15	28 519
-63	15	28 518
64	15	28 517
-64	15	28 516

Table SE129

Quantization level	Code length	Code
0	4	14
1	4	11
-1	4	10
2	4	7
-2	4	6
3	4	3
-3	4	2
4	5	31
-4	5	30
5	5	25
-5	5	24
6	5	17
-6	5	16
7	5	9
-7	5	8
8	5	1
-8	5	0
9	6	53
-9	6	52
10	6	37
-10	6	36
11	6	21
-11	6	20
12	6	5
-12	6	4
13	7	109
-13	7	108
14	7	77
-14	7	76
15	7	45
-15	7	44
16	7	13
-16	7	12
17	8	221
-17	8	220
18	8	157
-18	8	156
19	8	93
-19	8	92
20	8	29
-20	8	28
21	9	445
-21	9	444
22	9	317
-22	9	316
23	9	189
-23	9	188
24	9	61
-24	9	60
25	10	892
-25	10	639
26	10	637
-26	10	636
27	10	381
-27	10	380
28	10	125
-28	10	124
29	11	1 788
-29	11	1 787
30	11	1 276
-30	11	767
31	11	764
-31	11	255
32	11	252
-32	12	3 583
33	12	3 579
-33	12	3 578
34	12	2 555
-34	12	2 554
35	12	1 531
-35	12	1 530
36	12	507
-36	12	506
37	13	7 160
-37	13	7 147
38	13	7 144
-38	13	3 067
39	13	3 065
-39	13	3 064
40	13	1 017
-40	13	1 016
41	14	14 330
-41	14	14 329
42	14	14 291
-42	14	14 290
43	14	6 132
-43	14	2 039
44	14	2 038
-44	14	2 037
45	15	28 663
-45	15	28 662
46	15	28 585
-46	15	28 584
47	15	12 267
-47	15	12 266
48	15	4 073
-48	15	4 072
49	16	57 315
-49	16	57 314
50	16	57 313
-50	16	57 312
51	16	57 311
-51	16	57 310
52	16	57 309
-52	16	57 308
53	16	57 307
-53	16	57 306
54	16	57 305
-54	16	57 304
55	16	57 303
-55	16	57 302
56	16	57 301
-56	16	57 300
57	16	57 299
-57	16	57 298
58	16	57 297
-58	16	57 296
59	16	57 295
-59	16	57 294
60	16	57 293
-60	16	57 292
61	16	57 291
-61	16	57 290
62	16	57 289
-62	16	57 288
63	16	57 175
-63	16	57 174
64	16	57 173
-64	16	57 172

Table A129

Quantization level	Code length	Code
0	4	8
1	4	10
-1	4	9
2	4	0
-2	5	31
3	5	24
-3	5	23
4	5	12
-4	5	11
5	5	5
-5	5	4
6	6	60
-6	6	58
7	6	54
-7	6	53
8	6	45
-8	6	44
9	6	28
-9	6	27
10	6	19
-10	6	18
11	6	14
-11	6	13
12	6	6
-12	6	5
13	7	122
-13	7	119
14	7	113
-14	7	112
15	7	104
-15	7	103
16	7	100
-16	7	63
17	7	60
-17	7	59
18	7	52
-18	7	43
19	7	40
-19	7	35
20	7	32
-20	7	31
21	7	15
-21	7	14
22	8	247
-22	8	246

Quantization level	Code length	Code
23	8	231
-23	8	230
24	8	223
-24	8	222
25	8	211
-25	8	210
26	8	203
-26	8	202
27	8	123
-27	8	122
28	8	116
-28	8	107
29	8	84
-29	8	83
30	8	68
-30	8	67
31	8	60
-31	8	51
32	8	49
-32	8	48
33	8	17
-33	8	16
34	9	474
-34	9	473
35	9	458
-35	9	457
36	9	442
-36	9	441
37	9	411
-37	9	410
38	9	251
-38	9	250
39	9	248
-39	9	235
40	9	213
-40	9	212
41	9	170
-41	9	165
42	9	139
-42	9	138
43	9	132
-43	9	123
44	9	101
-44	9	100
45	9	37

Quantization level	Code length	Code
-45	9	36
46	10	950
-46	10	945
47	10	919
-47	10	918
48	10	912
-48	10	887
49	10	881
-49	10	880
50	10	818
-50	10	817
51	10	499
-51	10	498
52	10	469
-52	10	468
53	10	343
-53	10	342
54	10	329
-54	10	328
55	10	267
-55	10	266
56	10	245
-56	10	244
57	10	79
-57	10	78
58	10	77
-58	10	76
59	11	1 903
-59	11	1 902
60	11	1 889
-60	11	1 888
61	11	1 827
-61	11	1 826
62	11	1 773
-62	11	1 772
63	11	1 639
-63	11	1 638
64	11	1 633
-64	11	1 632

Table B129

Quantization level	Code length	Code
0	5	10
1	5	7
-1	5	6
2	5	4
-2	5	3
3	5	0
-3	6	63
4	6	60
-4	6	59
5	6	57
-5	6	56
6	6	53
-6	6	52
7	6	50
-7	6	49
8	6	46
-8	6	45
9	6	43
-9	6	42
10	6	39
-10	6	38
11	6	35
-11	6	34
12	6	32
-12	6	31
13	6	28
-13	6	27
14	6	25
-14	6	24
15	6	22
-15	6	19
16	6	16
-16	6	11
17	6	5
-17	6	4
18	7	125
-18	7	124
19	7	122
-19	7	117
20	7	110
-20	7	109
21	7	103
-21	7	102
22	7	96
-22	7	95
23	7	89
-23	7	88
24	7	81
-24	7	80
25	7	74
-25	7	73
26	7	66
-26	7	61
27	7	59
-27	7	58
28	7	52
-28	7	47
29	7	37
-29	7	36
30	7	21
-30	7	20
31	7	6
-31	7	5
32	8	247
-32	8	246
33	8	223
-33	8	222
34	8	217
-34	8	216
35	8	189
-35	8	188
36	8	166
-36	8	165
37	8	151
-37	8	150
38	8	144
-38	8	135
39	8	121
-39	8	120
40	8	106
-40	8	93
41	8	71
-41	8	70
42	8	68
-42	8	15
43	8	9
-43	8	8
44	9	466
-44	9	465
45	9	391
-45	9	390
46	9	388
-46	9	335
47	9	329
-47	9	328
48	9	269
-48	9	268
49	9	215
-49	9	214
50	9	184
-50	9	139
51	9	29
-51	9	28
52	10	934
-52	10	929
53	10	779
-53	10	778
54	10	668
-54	10	583
55	10	582
-55	10	581
56	10	371
-56	10	370
57	10	276
-57	11	1 871
58	11	1 857
-58	11	1 856
59	11	1 338
-59	11	1 161
60	11	1 160
-60	11	555
61	12	3 741
-61	12	3 740
62	12	2 678
-62	12	1 109
63	12	1 108
-63	13	5 359
64	14	10 717
-64	14	10 716

Table C129

Quantization level	Code length	Code
0	6	58
1	6	55
-1	6	54
2	6	52
-2	6	51
3	6	49
-3	6	48
4	6	46
-4	6	45
5	6	43
-5	6	42
6	6	40
-6	6	39
7	6	37
-7	6	36
8	6	34
-8	6	33
9	6	30
-9	6	29
10	6	27
-10	6	26
11	6	24
-11	6	23
12	6	21
-12	6	20
13	6	18
-13	6	17
14	6	14
-14	6	13
15	6	12
-15	6	11
16	6	8
-16	6	7
17	6	6
-17	6	5
18	6	3
-18	6	2
19	7	127
-19	7	126
20	7	124
-20	7	123
21	7	121
-21	7	120
22	7	118
-22	7	115
23	7	113
-23	7	112
24	7	106
-24	7	101
25	7	95
-25	7	94
26	7	88
-26	7	83
27	7	77
-27	7	76
28	7	70
-28	7	65
29	7	64
-29	7	63
30	7	56
-30	7	51
31	7	45
-31	7	44
32	7	39
-32	7	38
33	7	31
-33	7	30
34	7	20
-34	7	19
35	7	18
-35	7	9
36	7	3
-36	7	2
37	7	0
-37	8	251
38	8	245
-38	8	244
39	8	238
-39	8	229
40	8	215
-40	8	214
41	8	200
-41	8	179
42	8	165
-42	8	164
43	8	143
-43	8	142
44	8	124
-44	8	115
45	8	101
-45	8	100
46	8	66
-46	8	65
47	8	43
-47	8	42
48	8	17
-48	8	16
49	8	2
-49	9	501
50	9	479
-50	9	478
51	9	456
-51	9	403
52	9	357
-52	9	356
53	9	251
-53	9	250
54	9	228
-54	9	135
55	9	129
-55	9	128
56	9	6
-56	10	1 001
57	10	1 000
-57	10	915
58	10	805
-58	10	804
59	10	458
-59	10	269
60	10	268
-60	10	15
61	11	1 829
-61	11	1 828
62	11	918
-62	11	29
63	11	28
-63	12	1 839
64	13	3 677
-64	13	3 676

Table D129

Quantization level	Code length	Code
0	4	9
1	4	6
-1	4	5
2	4	2
-2	4	1
3	5	30
-3	5	29
4	5	26
-4	5	25
5	5	22
-5	5	21
6	5	16
-6	5	15
7	5	8
-7	5	7
8	5	0
-8	6	63
9	6	56
-9	6	55
10	6	48
-10	6	47
11	6	40
-11	6	35
12	6	28
-12	6	19
13	6	12
-13	6	3
14	7	124
-14	7	115
15	7	108
-15	7	99
16	7	92
-16	7	83
17	7	68
-17	7	59
18	7	36
-18	7	27
19	7	4
-19	8	251
20	8	228
-20	8	219
21	8	196
-21	8	187
22	8	164
-22	8	139

Quantization level	Code length	Code
23	8	116
-23	8	75
24	8	52
-24	8	11
25	9	501
-25	9	500
26	9	437
-26	9	436
27	9	373
-27	9	372
28	9	277
-28	9	276
29	9	149
-29	9	148
30	9	21
-30	9	20
31	10	917
-31	10	916
32	10	789
-32	10	788
33	10	661
-33	10	660
34	10	469
-34	10	468
35	10	214
-35	10	213
36	11	1 838
-36	11	1 837
37	11	1 582
37	11	1 581
38	11	1 326
-38	11	1 325
39	11	942
-39	11	941
40	11	431
-40	11	430
41	12	3 679
-41	12	3 678
42	12	3 167
-42	12	3 166
43	12	3 160
-43	12	2 655
44	12	2 648
-44	12	1 887
45	12	1 880

Quantization level	Code length	Code
-45	12	851
46	12	849
-46	12	848
47	13	7 346
-47	13	7 345
48	13	6 322
-48	13	5 309
49	13	3 773
-49	13	3 772
50	13	3 762
-50	13	1 701
51	14	14 695
-51	14	14 694
52	14	14 688
-52	14	12 647
53	14	10 617
-53	14	10 616
54	14	10 596
-54	14	7 527
55	14	3 401
-55	14	3 400
56	15	29 378
-56	15	25 293
57	15	21 195
-57	15	21 194
58	15	15 053
-58	15	15 052
59	16	58 759
-59	16	58 758
60	16	50 585
-60	16	50 584
61	16	42 399
-61	16	42 398
62	16	42 397
-62	16	42 396
63	16	42 395
-63	16	42 394
64	16	42 393
-64	16	42 392

Table E129

Quantization level	Code length	Code
0	5	12
1	5	11
-1	5	10
2	5	9
-2	5	8
3	5	7
-3	5	6
4	5	4
-4	5	3
5	5	2
-5	5	1
6	5	0
-6	6	63
7	6	61
-7	6	60
8	6	59
-8	6	58
9	6	56
-9	6	55
10	6	53
-10	6	52
11	6	51
-11	6	50
12	6	47
-12	6	46
13	6	45
-13	6	44
14	6	42
-14	6	41
15	6	38
-15	6	37
16	6	36
-16	6	35
17	6	32
-17	6	31
18	6	29
-18	6	28
19	6	26
-19	6	11
20	7	125
-20	7	124
21	7	109
-21	7	108
22	7	98
-22	7	97
23	7	87
-23	7	86
24	7	79
-24	7	78
25	7	68
-25	7	67
26	7	60
-26	7	55
27	7	21
-27	7	20
28	8	230
-28	8	229
29	8	198
-29	8	193
30	8	163
-30	8	162
31	8	139
-31	8	138
32	8	123
-32	8	122
33	8	108
-33	9	463
34	9	457
-34	9	456
35	9	385
-35	9	384
36	9	321
-36	9	320
37	9	266
-37	9	265
38	9	218
-38	10	925
39	10	798
-39	10	797
40	10	646
-40	10	645
41	10	535
-41	10	534
42	10	528
-42	10	439
43	11	1 848
-43	11	1 599
44	11	1 592
-44	11	1 295
45	11	1 288
-45	11	1 059
46	11	877
-46	11	876
47	12	3 197
-47	12	3 196
48	12	2 589
-48	12	2 588
49	12	2 117
-49	12	2 116
50	13	7 398
-50	13	7 397
51	13	6 374
-51	13	6 373
52	13	5 158
-52	13	5 157
53	14	14 799
-53	14	14 798
54	14	12 751
-54	14	12 750
55	14	10 318
-55	14	10 313
56	15	29 587
-56	15	29 586
57	15	29 584
-57	15	25 491
58	15	20 625
-58	15	20 624
59	16	59 171
-59	16	59 170
60	16	50 980
-60	16	41 277
61	16	50 981
-61	16	41 278
62	16	50 978
-62	16	41 279
63	16	50 979
-63	16	50 976
64	16	50 977
-64	16	41 276

Table F129

Quantization level	Code length	Code
0	6	56
1	6	55
-1	6	54
2	6	52
-2	6	51
3	6	50
-3	6	49
4	6	48
-4	6	47
5	6	46
-5	6	45
6	6	44
-6	6	43
7	6	41
-7	6	40
8	6	39
-8	6	38
9	6	36
-9	6	35
10	6	34
-10	6	33
11	6	31
-11	6	30
12	6	29
-12	6	28
13	6	26
-13	6	25
14	6	23
-14	6	22
15	6	21
-15	6	20
16	6	18
-16	6	17
17	6	15
-17	6	14
18	6	12
-18	6	11
19	6	9
-19	6	8
20	6	7
-20	6	6
21	6	3
-21	6	2
22	6	1
-22	6	0

Quantization level	Code length	Code
23	7	125
-23	7	124
24	7	123
-24	7	122
25	7	120
-25	7	119
26	7	116
-26	7	115
27	7	114
-27	7	107
28	7	84
-28	7	75
29	7	65
-29	7	64
30	7	54
-30	7	49
31	7	39
-31	7	38
32	7	27
-32	7	26
33	7	20
-33	7	11
34	7	10
-34	7	9
35	8	254
-35	8	253
36	8	243
-36	8	242
37	8	235
-37	8	234
38	8	213
-38	8	212
39	8	149
-39	8	148
40	8	110
-40	8	97
41	8	66
-41	8	65
42	8	43
-42	8	42
43	8	16
-43	9	511
44	9	505
-44	9	504
45	9	474

Quantization level	Code length	Code
-45	9	473
46	9	343
-46	9	342
47	9	340
-47	9	223
48	9	192
-48	9	135
49	9	129
-49	9	128
50	9	34
-50	10	1 021
51	10	951
-51	10	950
52	10	944
-52	10	683
53	10	445
-53	10	444
54	10	269
-54	10	268
55	10	71
-55	10	70
56	11	2 040
-56	11	1 891
57	11	1 364
-57	11	775
58	11	774
-58	11	773
59	12	4 083
-59	12	4 082
60	12	3 780
-60	12	2 731
61	12	1 545
-61	12	1 544
62	13	7 562
-62	13	5 461
63	13	5 460
-63	14	15 127
64	15	30 253
-64	15	30 252

Table G129

Quantization level	Code length	Code
0	4	0
1	5	29
-1	5	28
2	5	25
-2	5	24
3	5	21
-3	5	20
4	5	17
-4	5	16
5	5	13
-5	5	12
6	5	9
-6	5	8
7	5	5
-7	5	4
8	6	63
-8	6	62
9	6	55
-9	6	54
10	6	47
-10	6	46
11	6	39
-11	6	38
12	6	31
-12	6	30
13	6	23
-13	6	22
14	6	15
-14	6	14
15	6	7
-15	6	6
16	7	123
-16	7	122
17	7	107
-17	7	106
18	7	91
-18	7	90
19	7	75
-19	7	74
20	7	59
-20	7	58
21	7	43
-21	7	42
22	7	27
-22	7	26
23	7	11
-23	7	10
24	7	8
-24	8	243
25	8	240
-25	8	211
26	8	208
-26	8	179
27	8	176
-27	8	147
28	8	144
-28	8	115
29	8	112
-29	8	83
30	8	80
-30	8	51
31	8	48
-31	8	19
32	9	484
-32	9	483
33	9	421
-33	9	420
34	9	357
-34	9	356
35	9	293
-35	9	292
36	9	229
-36	9	228
37	9	226
-37	9	165
38	9	162
-38	9	101
39	9	98
-39	9	37
40	10	970
-40	10	965
41	10	839
-41	10	838
42	10	711
-42	10	710
43	10	708
-43	10	583
44	10	580
-44	10	455
45	10	329
-45	10	328
46	10	201
-46	10	200
47	10	198
-47	10	73
48	11	1 942
-48	11	1 929
49	11	1 675
-49	11	1 674
50	11	1 672
-50	11	1 419
51	11	1 165
-51	11	1 164
52	11	1 162
-52	11	909
53	11	655
-53	11	654
54	11	652
-54	11	399
55	11	145
-55	11	144
56	12	3 886
-56	12	3 857
57	12	3 347
-57	12	3 346
58	12	2 837
-58	12	2 836
59	12	2 327
-59	12	2 326
60	12	1 817
-60	12	1 816
61	12	1 307
-61	12	1 306
62	12	797
-62	12	796
63	13	7 775
-63	13	7 774
64	13	7 713
-64	13	7 712

D.6 Block Code Books

D.6.1 3 Levels

Table V.3: 3-level 4-element 7-bit Block Code Book

Level index	Code for 1st element
-1	0
0	1
1	2
Level index	**Code for 2nd element**
-1	0
0	3
1	6
Level index	**Code for 3rd element**
-1	0
0	9
1	18
Level index	**Code for 4th element**
-1	0
0	27
1	54

D.6.2 5 Levels

Table V.5: 5-level 4-element 10-bit Block Code Book

Level index	Code for 1st element
-2	0
-1	1
0	2
1	3
2	4
Level index	**Code for 2nd element**
-2	0
-1	5
0	10
1	15
2	20
Level index	**Code for 3rd element**
-2	0
-1	25
0	50
1	75
2	100
Level index	**Code for 4th element**
-2	0
-1	125
0	250
1	375
2	500

D.6.3 7 Levels

Table V.7: 7-level 4-element 12-bit Block Code Book

Level index	Code for 1st element
-3	0
-2	1
-1	2
0	3
1	4
2	5
3	6
Level index	**Code for 2nd element**
-3	0
-2	7
-1	14
0	21
1	28
2	35
3	42
Level index	**Code for 3rd element**
-3	0
-2	49
-1	98
0	47
1	196
2	245
3	294
Level index	**Code for 4th element**
-3	0
-2	343
-1	686
0	1 029
1	1 372
2	1 715
3	2 058

D.6.4 9 Levels

Table V.9: 9-level 4-element 13-bit Block Code Book

Level index	Code for 1st element
-4	0
-3	1
-2	2
-1	3
0	4
1	5
2	6
3	7
4	8
Level index	**Code for 2nd element**
-4	0
-3	9
-2	18
-1	27
0	36
1	45
2	54
3	63
4	72

Level index	Code for 3rd element
-4	0
-3	81
-2	162
-1	243
0	324
1	405
2	486
3	567
4	648
Level index	**Code for 4th element**
-4	0
-3	729
-2	1 458
-1	2 187
0	2 916
1	3 645
2	4 374
3	5 103
4	5 832

D.6.5 13 Levels

Table V.13: 13-level 4-element 15-bit block

Level index	Code for 1st element
-6	0
-5	1
-4	2
-3	3
-2	4
-1	5
0	6
1	7
2	8
3	9
4	10
5	11
6	12
Level index	**Code for 2nd element**
-6	0
-5	13
-4	26
-3	39
-2	52
-1	65
0	78
1	91
2	104
3	117
4	130
5	143
6	156

Level index	Code for 3rd element
-6	0
-5	169
-4	338
-3	507
-2	676
-1	845
0	1 014
1	1 183
2	1 352
3	1 521
4	1 690
5	1 859
6	2 028
Level index	**Code for 4th element**
-6	0
-5	2 197
-4	4 394
-3	6 591
-2	8 788
-1	10 985
0	13 182
1	15 379
2	17 576
3	19 773
4	21 970
5	24 167
6	26 364

D.6.6 17 Levels

Table V.17: 17-level 4-element 17-bit Block Code Book

Level index	Code for 1st element
-8	0
-7	1
-6	2
-5	3
-4	4
-3	5
-2	6
-1	7
0	8
1	9
2	10
3	11
4	12
5	13
6	14
7	15
8	16

Level index	Code for 2nd element
-8	0
-7	17
-6	34
-5	51
-4	68
-3	85
-2	102
-1	119
0	136
1	153
2	170
3	187
4	204
5	221
6	238
7	255
8	272

Level index	Code for 3rd element
-8	0
-7	289
-6	578
-5	867
-4	1 156
-3	1 445
-2	1 734
-1	2 023
0	2 312
1	2 601
2	2 890
3	3 179
4	3 468
5	3 757
6	4 046
7	4 335
8	4 624

Level index	Code for 4th element
-8	0
-7	4 913
-6	9 826
-5	14 739
-4	19 652
-3	24 565
-2	29 478
-1	34 391
0	39 304
1	44 217
2	49 130
3	54 043
4	58 956
5	63 869
6	68 782
7	73 695
8	78 608

D.6.7 25 Levels

Table V.25: 25-level 4-element 19-bit Block Code Book

Level index	Code for 1st element
-12	0
-11	1
-10	2
-9	3
-8	4
-7	5
-6	6
-5	7
-4	8
-3	9
-2	10
-1	11
0	12
1	13
2	14
3	15
4	16
5	17
6	18
7	19
8	20
9	21
10	22
11	23
12	24

Level index	Code for 2nd element
-12	0
-11	25
-10	50
-9	75
-8	100
-7	125
-6	150
-5	175
-4	200
-3	225
-2	250
-1	275
0	300
1	325
2	350
3	375
4	400
5	425
6	450
7	475
8	500
9	525
10	550
11	575
12	600

Level index	Code for 3rd element
-12	0
-11	625
-10	1 250
-9	1 875
-8	2 500
-7	3 125
-6	3 750
-5	4 375
-4	5 000
-3	5 625
-2	6 250
-1	6 875
0	7 500
1	8 125
2	8 750
3	9 375
4	10 000
5	10 625
6	11 250
7	11 875
8	12 500
9	13 125
10	13 750
11	14 375
12	15 000

Level index	Code for 4th element
-12	0
-11	15 625
-10	31 250
-9	46 875
-8	62 500
-7	78 125
-6	93 750
-5	109 375
-4	125 000
-3	140 625
-2	156 250
-1	171 875
0	187 500
1	203 125
2	218 750
3	234 375
4	250 000
5	265 625
6	281 250
7	296 875
8	312 500
9	328 125
10	343 750
11	359 375
12	375 000

D.7 Interpolation FIR

Index	FIR - 2 x Interpolation	FIR - 4 x Interpolation
0	3,305240000e-06	2,107630000e-06
1	-1,095500000e-07	1,094810000e-05
2	-1,133348000e-05	2,290807000e-05
3	-5,509460000e-06	2,839700000e-05
4	2,381930000e-05	1,428398000e-05
5	2,278368000e-05	-2,752976000e-05
6	-3,684078000e-05	-8,951150000e-05
7	-5,886791000e-05	-1,427962100e-04
8	4,053684000e-05	-1,435831500e-04
9	1,186829100e-04	-5,408613000e-05
10	-1,809484000e-05	1,283221800e-04
11	-2,002544900e-04	3,488978300e-04
12	-5,299183000e-05	4,982510600e-04
13	2,892986200e-04	4,505801800e-04
14	1,963655800e-04	1,306004900e-04
15	-3,546474000e-04	-4,182235900e-04
16	-4,285478200e-04	-1,004001470e-03
17	3,466888200e-04	-1,328857730e-03
18	7,476581400e-04	-1,108668630e-03
19	-2,011064500e-04	-2,332188200e-04
20	-1,121123670e-03	1,106532170e-03
21	-1,503691300e-04	2,405677920e-03
22	1,475675030e-03	2,994750860e-03
23	7,618262300e-04	2,327672210e-03
24	-1,693736650e-03	3,030619100e-04
25	-1,649267160e-03	-2,537531080e-03
26	1,620259490e-03	-5,075346680e-03
27	2,764807080e-03	-5,991244690e-03
28	-1,082837000e-03	-4,355599640e-03
29	-3,974851220e-03	-1,972305800e-04
30	-7,440893000e-05	5,237236150e-03
31	5,049238450e-03	9,746226480e-03
32	1,945910740e-03	1,096914243e-02
33	-5,668033380e-03	7,467648480e-03
34	-4,514896780e-03	-3,564674400e-04
35	5,450623580e-03	-9,986897000e-03
36	7,607854900e-03	-1,744846255e-02
37	-4,008148330e-03	-1,880371012e-02
38	-1,086365897e-02	-1,198318321e-02
39	1,015614490e-03	1,828493200e-03
40	1,372703910e-02	1,799243502e-02
41	3,704760920e-03	2,975338697e-02
42	-1,547267288e-02	3,083455376e-02
43	-1,010103151e-02	1,837555505e-02
44	1,526044402e-02	-5,042277280e-03
45	1,782309450e-02	-3,137993813e-02
46	-1,221452747e-02	-4,954963177e-02
47	-2,617896535e-02	-4,967092723e-02
48	5,509705280e-03	-2,767589502e-02
49	3,411839902e-02	1,166744903e-02
50	5,563403950e-03	5,492079630e-02
51	-4,023005813e-02	8,387579024e-02
52	-2,160109766e-02	8,227037638e-02
53	4,270342737e-02	4,309020936e-02
54	4,324966297e-02	-2,637432516e-02
55	-3,911506757e-02	-1,040880680e-01
56	-7,177370787e-02	-1,583629698e-01
57	2,557834797e-02	-1,573987603e-01
58	1,109383851e-01	-8,037899435e-02
59	6,777029020e-03	7,367454469e-02

Index	FIR - 2 x Interpolation	FIR - 4 x Interpolation
60	-1,752302796e-01	2,826547325e-01
61	-9,289701283e-02	5,053876638e-01
62	3,630628884e-01	6,921411753e-01
63	8,234865069e-01	7,985475659e-01
64	8,234865069e-01	7,985475659e-01
65	3,630628884e-01	6,921411753e-01
66	-9,289701283e-02	5,053876638e-01
67	-1,752302796e-01	2,826547325e-01
68	6,777029020e-03	7,367454469e-02
69	1,109383851e-01	-8,037899435e-02
70	2,557834797e-02	-1,573987603e-01
71	-7,177370787e-02	-1,583629698e-01
72	-3,911506757e-02	-1,040880680e-01
73	4,324966297e-02	-2,637432516e-02
74	4,270342737e-02	4,309020936e-02
75	-2,160109766e-02	8,227037638e-02
76	-4,023005813e-02	8,387579024e-02
77	5,563403950e-03	5,492079630e-02
78	3,411839902e-02	1,166744903e-02
79	5,509705280e-03	-2,767589502e-02
80	-2,617896535e-02	-4,967092723e-02
81	-1,221452747e-02	-4,954963177e-02
82	1,782309450e-02	-3,137993813e-02
83	1,526044402e-02	-5,042277280e-03
84	-1,010103151e-02	1,837555505e-02
85	-1,547267288e-02	3,083455376e-02
86	3,704760920e-03	2,975338697e-02
87	1,372703910e-02	1,799243502e-02
88	1,015614490e-03	1,828493200e-03
89	-1,086365897e-02	-1,198318321e-02
90	-4,008148330e-03	-1,880371012e-02
91	7,607854900e-03	-1,744846255e-02
92	5,450623580e-03	-9,986897000e-03
93	-4,514896780e-03	-3,564674400e-04
94	-5,668033380e-03	7,467648480e-03
95	1,945910740e-03	1,096914243e-02
96	5,049238450e-03	9,746226480e-03
97	-7,440893000e-05	5,237236150e-03
98	-3,974851220e-03	-1,972305800e-04
99	-1,082837000e-03	-4,355599640e-03
100	2,764807080e-03	-5,991244690e-03
101	1,620259490e-03	-5,075346680e-03
102	-1,649267160e-03	-2,537531080e-03
103	-1,693736650e-03	3,030619100e-04
104	7,618262300e-04	2,327672210e-03
105	1,475675030e-03	2,994750860e-03
106	-1,503691300e-04	2,405677920e-03
107	-1,121123670e-03	1,106532170e-03
108	-2,011064500e-04	-2,332188200e-04
109	7,476581400e-04	-1,108668630e-03
110	3,466888200e-04	-1,328857730e-03
111	-4,285478200e-04	-1,004001470e-03
112	-3,546474000e-04	-4,182235900e-04
113	1,963655800e-04	1,306004900e-04
114	2,892986200e-04	4,505801800e-04
115	-5,299183000e-05	4,982510600e-04
116	-2,002544900e-04	3,488978300e-04
117	-1,809484000e-05	1,283221800e-04
118	1,186829100e-04	-5,408613000e-05
119	4,053684000e-05	-1,435831500e-04
120	-5,886791000e-05	-1,427962100e-04
121	-3,684078000e-05	-8,951150000e-05
122	2,278368000e-05	-2,752976000e-05

Index	FIR - 2 x Interpolation	FIR - 4 x Interpolation
123	2,381930000e-05	1,428398000e-05
124	-5,509460000e-06	2,839700000e-05
125	-1,133348000e-05	2,290807000e-05
126	-1,095500000e-07	1,094810000e-05
127	3,305240000e-06	2,107630000e-06

D.8 32-Band Interpolation and LFE Interpolation FIR

Index	32-Band Interpolation FIR		LFE Interpolation FIR	
	Perfect Reconstruction	Non-Perfect Reconstruction	64 x Interpolation	128 x Interpolation
0	1,140033200e-10	-1,390191784e-07	2,658434387e-04	5,316857100e-04
1	7,138742100e-11	-1,693738625e-07	8,179365250e-05	1,635869100e-04
2	-8,358679600e-09	-2,030677564e-07	9,439323912e-05	1,887860900e-04
3	-2,529296600e-08	-2,404238444e-07	1,082170274e-04	2,164336300e-04
4	-9,130198800e-08	-2,818143514e-07	1,233371440e-04	2,466738200e-04
5	-2,771560000e-07	-3,276689142e-07	1,397485757e-04	2,794966000e-04
6	-5,746147600e-07	-3,784752209e-07	1,575958013e-04	3,151909600e-04
7	-3,712986200e-07	-4,347855338e-07	1,769922383e-04	3,539837500e-04
8	-4,468735700e-07	-4,972276315e-07	1,981738606e-04	3,963469100e-04
9	-5,697322600e-07	-5,665120852e-07	2,211847313e-04	4,423685900e-04
10	-6,300390500e-07	-6,434325428e-07	2,460231190e-04	4,920452500e-04
11	-6,677818900e-07	-7,288739425e-07	2,726115927e-04	5,452220800e-04
12	-6,770656500e-07	-8,238164355e-07	3,013863170e-04	6,027714100e-04
13	-6,601852300e-07	-9,293416952e-07	3,328395542e-04	6,656776500e-04
14	-6,193701600e-07	-1,046637067e-06	3,658991191e-04	7,317967800e-04
15	-5,586146700e-07	-1,176999604e-06	4,018281470e-04	8,036546600e-04
16	7,034745600e-07	-1,321840614e-06	4,401875485e-04	8,803732300e-04
17	8,348606100e-07	-1,482681114e-06	4,812776169e-04	9,625531400e-04
18	9,544782800e-07	-1,661159786e-06	5,252459669e-04	1,050489840e-03
19	1,052683900e-06	-1,859034001e-06	5,721592461e-04	1,144316160e-03
20	1,119829700e-06	-2,078171747e-06	6,222130032e-04	1,244423330e-03
21	1,144180200e-06	-2,320550948e-06	6,755515351e-04	1,351100280e-03
22	1,124542400e-06	-2,588257530e-06	7,324148901e-04	1,464826870e-03
23	9,822894700e-07	-2,883470643e-06	7,928516716e-04	1,585700080e-03
24	8,920065800e-07	-3,208459020e-06	8,570110658e-04	1,714018640e-03
25	1,560941800e-06	-3,565570978e-06	9,251192096e-04	1,850234690e-03
26	8,454480100e-07	-3,957220997e-06	9,974770946e-04	1,994950230e-03
27	3,167104300e-07	-4,385879038e-06	1,073930296e-03	2,147856400e-03
28	1,028149000e-07	-4,854050530e-06	1,155023579e-03	2,310042500e-03
29	4,147967800e-08	-5,364252502e-06	1,240676851e-03	2,481348810e-03
30	-6,821591800e-10	-5,918994248e-06	1,331258914e-03	2,662512240e-03
31	-1,611726200e-09	-6,520755960e-06	1,426893868e-03	2,853781920e-03
32	-2,668096400e-09	-7,171964626e-06	1,527829794e-03	3,055653300e-03
33	-3,377455500e-09	-7,874960829e-06	1,634211512e-03	3,268416510e-03
34	6,820855300e-09	-8,631964192e-06	1,746327500e-03	3,492647550e-03
35	3,715261200e-09	-9,445050637e-06	1,864377526e-03	3,728747140e-03
36	1,643020800e-08	-1,031611009e-05	1,988604199e-03	3,977200480e-03
37	1,007547900e-07	-1,124680875e-05	2,119151875e-03	4,238294900e-03
38	2,448299500e-07	-1,223855270e-05	2,256359672e-03	4,512710030e-03
39	1,306777300e-06	-1,329243969e-05	2,400433412e-03	4,800856580e-03
40	1,904890000e-06	-1,440921824e-05	2,551567042e-03	5,103122910e-03
41	2,555774300e-06	-1,558924305e-05	2,710093278e-03	5,420174920e-03
42	3,253336000e-06	-1,683242772e-05	2,876190469e-03	5,752369300e-03
43	3,953604500e-06	-1,813820381e-05	3,050152911e-03	6,100293250e-03
44	4,617880200e-06	-1,950545993e-05	3,232272575e-03	6,464532110e-03
45	5,210775600e-06	-2,093250441e-05	3,422776936e-03	6,845539900e-03
46	5,696789700e-06	-2,241701623e-05	3,621967277e-03	7,243919190e-03
47	6,046428700e-06	-2,395598858e-05	3,830091329e-03	7,660165890e-03
48	7,614387900e-06	-2,554569073e-05	4,047499038e-03	8,094980380e-03
49	7,678809700e-06	-2,718161704e-05	4,274417181e-03	8,548815730e-03

Index	32-Band Interpolation FIR		LFE Interpolation FIR	
	Perfect Reconstruction	Non-Perfect Reconstruction	64 x Interpolation	128 x Interpolation
50	7,533601500e-06	-2,885844333e-05	4,511159845e-03	9,022301060e-03
51	7,179758900e-06	-3,056998685e-05	4,758012015e-03	9,516004470e-03
52	6,629955000e-06	-3,230916263e-05	5,015311297e-03	1,003060210e-02
53	5,908209500e-06	-3,406793985e-05	5,283284001e-03	1,056654565e-02
54	5,044609200e-06	-3,583733633e-05	5,562345497e-03	1,112466771e-02
55	4,187209700e-06	-3,760734762e-05	5,852684379e-03	1,170534454e-02
56	3,139397100e-06	-3,936696885e-05	6,154712290e-03	1,230939943e-02
57	6,650809100e-07	-4,110412556e-05	6,468691397e-03	1,293735672e-02
58	3,073465500e-07	-4,280570283e-05	6,794991903e-03	1,358995494e-02
59	5,699348500e-08	-4,445751256e-05	7,133882027e-03	1,426773332e-02
60	1,510238900e-08	-4,604430433e-05	7,485736627e-03	1,497144438e-02
61	3,384827600e-08	-4,754976908e-05	7,850865833e-03	1,570170000e-02
62	-3,227406600e-08	-4,895655002e-05	8,229630999e-03	1,645922661e-02
63	-3,772031200e-08	-5,024627535e-05	8,622321300e-03	1,724460535e-02
64	8,454083600e-08	5,139957648e-05	9,029330686e-03	1,805862412e-02
65	6,479789100e-08	5,239612074e-05	9,450953454e-03	1,890186779e-02
66	1,236415900e-06	5,321469871e-05	9,887560271e-03	1,977507770e-02
67	2,480143600e-06	5,383323878e-05	1,033949479e-02	2,067894675e-02
68	3,694976800e-06	5,422891263e-05	1,080708485e-02	2,161412500e-02
69	3,742137100e-06	5,437819709e-05	1,129068248e-02	2,258131653e-02
70	3,262621300e-06	5,425697600e-05	1,179065090e-02	2,358125709e-02
71	7,476824700e-06	5,384063843e-05	1,230732165e-02	2,461459488e-02
72	9,321632700e-06	5,310418419e-05	1,284105983e-02	2,568206564e-02
73	1,121856000e-05	5,202236207e-05	1,339218579e-02	2,678431384e-02
74	1,317522400e-05	5,056979353e-05	1,396108977e-02	2,792212367e-02
75	1,505747500e-05	4,872112549e-05	1,454808749e-02	2,909611352e-02
76	1,676702500e-05	4,645117951e-05	1,515355054e-02	3,030703776e-02
77	1,819741000e-05	4,373511547e-05	1,577781141e-02	3,155555204e-02
78	1,925789500e-05	4,054862075e-05	1,642123051e-02	3,284239396e-02
79	1,987389300e-05	3,686808850e-05	1,708412915e-02	3,416819125e-02
80	-3,076839000e-05	3,267079956e-05	1,776690222e-02	3,553372994e-02
81	-3,254459900e-05	2,793515523e-05	1,846982725e-02	3,693958372e-02
82	-3,367812600e-05	2,264085742e-05	1,919330470e-02	3,838652745e-02
83	-3,411568400e-05	1,676913780e-05	1,993762329e-02	3,987516090e-02
84	-3,382472000e-05	1,030297699e-05	2,070316114e-02	4,140623659e-02
85	-3,280414400e-05	3,227306706e-06	2,149021253e-02	4,298033938e-02
86	-3,109003600e-05	-4,470633485e-06	2,229913883e-02	4,459818453e-02
87	-2,861654300e-05	-1,280130618e-05	2,313023806e-02	4,626038298e-02
88	-2,571454500e-05	-2,177240640e-05	2,398385666e-02	4,796761274e-02
89	-1,870056200e-05	-3,138873581e-05	2,486028522e-02	4,972046614e-02
90	-1,771374800e-05	-4,165195787e-05	2,575986087e-02	5,151961371e-02
91	-1,568432200e-05	-5,256036457e-05	2,668286115e-02	5,336561054e-02
92	-1,128458200e-05	-6,410864444e-05	2,762960829e-02	5,525910854e-02
93	-6,805568100e-06	-7,628766616e-05	2,860039286e-02	5,720067024e-02
94	-5,671807300e-07	-8,908427117e-05	2,959549613e-02	5,919086933e-02
95	-9,974569000e-07	-1,024810626e-04	3,061520495e-02	6,123027951e-02
96	-1,466421500e-06	-1,164562127e-04	3,165979683e-02	6,331945211e-02
97	-1,846174800e-06	-1,309833024e-04	3,272953629e-02	6,545893103e-02
98	7,763173700e-08	-1,460311323e-04	3,382468969e-02	6,764923781e-02
99	1,809597500e-06	-1,615635992e-04	3,494550660e-02	6,989086419e-02
100	4,157326000e-06	-1,775395358e-04	3,609224036e-02	7,218432426e-02
101	7,240269200e-06	-1,939126523e-04	3,726511076e-02	7,453006506e-02
102	1,073666400e-05	-2,106313768e-04	3,846437484e-02	7,692859322e-02
103	2,089583800e-05	-2,276388550e-04	3,969023004e-02	7,938029617e-02
104	2,647159500e-05	-2,448728774e-04	4,094288871e-02	8,188561350e-02
105	3,196094400e-05	-2,622658503e-04	4,222255200e-02	8,444493264e-02
106	3,698112500e-05	-2,797449124e-04	4,352942482e-02	8,705867827e-02
107	4,149260300e-05	-2,972317743e-04	4,486365616e-02	8,972713351e-02
108	4,534151200e-05	-3,146430245e-04	4,622544348e-02	9,245070815e-02
109	4,846834800e-05	-3,318900708e-04	4,761491716e-02	9,522963315e-02
110	5,081695700e-05	-3,488793736e-04	4,903224111e-02	9,806428105e-02
111	5,236303900e-05	-3,655125911e-04	5,047753453e-02	1,009548605e-01
112	3,803557300e-06	-3,816867538e-04	5,195093155e-02	1,039016470e-01

Index	32-Band Interpolation FIR		LFE Interpolation FIR	
	Perfect Reconstruction	Non-Perfect Reconstruction	64 x Interpolation	128 x Interpolation
113	7,916183300e-06	-3,972945851e-04	5,345252529e-02	1,069048345e-01
114	1,191309700e-05	-4,122247046e-04	5,498242006e-02	1,099646092e-01
115	1,561346600e-05	-4,263620067e-04	5,654069409e-02	1,130811572e-01
116	1,881671400e-05	-4,395879805e-04	5,812742189e-02	1,162546203e-01
117	2,131957100e-05	-4,517810594e-04	5,974265561e-02	1,194850579e-01
118	2,295038200e-05	-4,628172028e-04	6,138643622e-02	1,227726117e-01
119	2,354812700e-05	-4,725702747e-04	6,305878609e-02	1,261173040e-01
120	2,291622100e-05	-4,809123348e-04	6,475970894e-02	1,295191795e-01
121	2,497457200e-05	-4,877146275e-04	6,648923457e-02	1,329781860e-01
122	1,979628700e-05	-4,928477574e-04	6,824731827e-02	1,364943385e-01
123	1,390508100e-05	-4,961824161e-04	7,003392279e-02	1,400675476e-01
124	7,179248900e-06	-4,975944757e-04	7,184901088e-02	1,436977387e-01
125	-1,614022200e-07	-4,969481961e-04	7,369252294e-02	1,473847479e-01
126	-1,518084500e-05	-4,941228544e-04	7,556436211e-02	1,511284113e-01
127	-1,610369300e-05	-4,889960401e-04	7,746443897e-02	1,549285650e-01
128	1,994364800e-05	4,814492422e-04	7,939263433e-02	1,587849557e-01
129	1,774116500e-05	4,713678791e-04	8,134882897e-02	1,626973301e-01
130	4,511232400e-05	4,586426076e-04	8,333285898e-02	1,666653752e-01
131	5,311715600e-05	4,431701091e-04	8,534456789e-02	1,706887931e-01
132	6,144976200e-05	4,248536134e-04	8,738376945e-02	1,747671962e-01
133	7,052899300e-05	4,036037717e-04	8,945026249e-02	1,789001823e-01
134	7,984114900e-05	3,793396754e-04	9,154383838e-02	1,830873191e-01
135	8,597821200e-05	3,519894381e-04	9,366425127e-02	1,873281151e-01
136	9,341758200e-05	3,214911267e-04	9,581124038e-02	1,916220933e-01
137	1,002681400e-04	2,877934603e-04	9,798453748e-02	1,959686577e-01
138	1,064814700e-04	2,508567995e-04	1,001838669e-01	2,003673166e-01
139	1,119841200e-04	2,106537577e-04	1,024089083e-01	2,048173845e-01
140	1,165901700e-04	1,671699720e-04	1,046593264e-01	2,093182206e-01
141	1,202018700e-04	1,204049113e-04	1,069347933e-01	2,138691545e-01
142	1,226936800e-04	7,037253090e-05	1,092349365e-01	2,184694260e-01
143	1,237377900e-04	1,710198012e-05	1,115593687e-01	2,231182903e-01
144	1,200453700e-04	-3,936182839e-05	1,139076948e-01	2,278149277e-01
145	1,185602000e-04	-9,895755647e-05	1,162794977e-01	2,325585187e-01
146	1,152534400e-04	-1,616069785e-04	1,186743453e-01	2,373482138e-01
147	1,097435100e-04	-2,272142592e-04	1,210917681e-01	2,421830446e-01
148	1,018237000e-04	-2,956659591e-04	1,235313043e-01	2,470620573e-01
149	9,130172200e-05	-3,668301215e-04	1,259924471e-01	2,519843280e-01
150	7,793692700e-05	-4,405563814e-04	1,284746826e-01	2,569487989e-01
151	6,157321800e-05	-5,166754709e-04	1,309774816e-01	2,619544268e-01
152	4,214289700e-05	-5,949990009e-04	1,335003078e-01	2,670000792e-01
153	2,010055900e-05	-6,753197522e-04	1,360425949e-01	2,720846236e-01
154	-6,512868000e-06	-7,574109477e-04	1,386037618e-01	2,772069275e-01
155	-3,623958500e-05	-8,410271257e-04	1,411831975e-01	2,823657692e-01
156	-6,898332300e-05	-9,259034996e-04	1,437802613e-01	2,875599265e-01
157	-1,052143400e-04	-1,011756598e-03	1,463943720e-01	2,927881181e-01
158	-1,311540500e-04	-1,098284614e-03	1,490248144e-01	2,980490029e-01
159	-1,772621900e-04	-1,185167348e-03	1,516709626e-01	3,033412695e-01
160	-2,231129500e-04	-1,272067428e-03	1,543320864e-01	3,086635172e-01
161	-2,678985000e-04	-1,358630019e-03	1,570075154e-01	3,140144050e-01
162	-3,353960600e-04	-1,444484224e-03	1,596965194e-01	3,193923831e-01
163	-3,909221300e-04	-1,529243193e-03	1,623983532e-01	3,247960210e-01
164	-4,488403900e-04	-1,612505526e-03	1,651122719e-01	3,302238286e-01
165	-5,091327500e-04	-1,693855622e-03	1,678375006e-01	3,356742859e-01
166	-5,717321000e-04	-1,772865304e-03	1,705732346e-01	3,411457539e-01
167	-6,360244700e-04	-1,849094522e-03	1,733186990e-01	3,466366828e-01
168	-7,021067600e-04	-1,922092517e-03	1,760730892e-01	3,521454632e-01
169	-7,695597500e-04	-1,991399564e-03	1,788355410e-01	3,576703668e-01
170	-8,380918900e-04	-2,056547208e-03	1,816052496e-01	3,632097244e-01
171	-9,072555100e-04	-2,117061289e-03	1,843813360e-01	3,687619269e-01
172	-9,767158300e-04	-2,172462177e-03	1,871629506e-01	3,743250966e-01
173	-1,045985500e-03	-2,222266514e-03	1,899491698e-01	3,798975349e-01
174	-1,114606900e-03	-2,265989315e-03	1,927391142e-01	3,854774535e-01
175	-1,182107000e-03	-2,303145360e-03	1,955319196e-01	3,910630047e-01

Index	32-Band Interpolation FIR		LFE Interpolation FIR	
	Perfect Reconstruction	Non-Perfect Reconstruction	64 x Interpolation	128 x Interpolation
176	-1,251459700e-03	-2,333251061e-03	1,983266175e-01	3,966524303e-01
177	-1,314813200e-03	-2,355825622e-03	2,011223286e-01	4,022437930e-01
178	-1,375058300e-03	-2,370394068e-03	2,039180547e-01	4,078352153e-01
179	-1,431717500e-03	-2,376487479e-03	2,067128718e-01	4,134248793e-01
180	-1,484159500e-03	-2,373647178e-03	2,095058411e-01	4,190107882e-01
181	-1,531686400e-03	-2,361423569e-03	2,122959495e-01	4,245910645e-01
182	-1,573715600e-03	-2,339380793e-03	2,150822729e-01	4,301636219e-01
183	-1,609496400e-03	-2,307097195e-03	2,178637981e-01	4,357266724e-01
184	-1,638393400e-03	-2,264167881e-03	2,206395119e-01	4,412781000e-01
185	-1,659751400e-03	-2,210205887e-03	2,234084606e-01	4,468160272e-01
186	-1,672691700e-03	-2,144844970e-03	2,261696160e-01	4,523383081e-01
187	-1,676540900e-03	-2,067740774e-03	2,289219648e-01	4,578429461e-01
188	-1,670887400e-03	-1,978572691e-03	2,316644788e-01	4,633280039e-01
189	-1,654649800e-03	-1,877046190e-03	2,343961596e-01	4,687913656e-01
190	-1,632849400e-03	-1,762894331e-03	2,371159792e-01	4,742309451e-01
191	-1,592423900e-03	-1,635878929e-03	2,398228943e-01	4,796448052e-01
192	1,541196600e-03	1,495792647e-03	2,425158769e-01	4,850307405e-01
193	1,478566700e-03	1,342460280e-03	2,451938838e-01	4,903867543e-01
194	1,394017000e-03	1,175740734e-03	2,478559017e-01	4,957108200e-01
195	1,301623400e-03	9,955273708e-04	2,505008876e-01	5,010007620e-01
196	1,194737700e-03	8,017504588e-04	2,531278133e-01	5,062545538e-01
197	1,072608600e-03	5,943773431e-04	2,557355762e-01	5,114701390e-01
198	9,349224800e-04	3,734139318e-04	2,583232224e-01	5,166453719e-01
199	7,810380900e-04	1,389056415e-04	2,608896792e-01	5,217782855e-01
200	6,109076600e-04	-1,090620208e-04	2,634339035e-01	5,268667936e-01
201	4,241331700e-04	-3,703625989e-04	2,659549415e-01	5,319088101e-01
202	2,204804700e-04	-6,448282511e-04	2,684516609e-01	5,369022489e-01
203	-2,272228400e-07	-9,322494152e-04	2,709231377e-01	5,418450832e-01
204	-2,380696500e-04	-1,232374110e-03	2,733682692e-01	5,467353463e-01
205	-4,930996000e-04	-1,544908970e-03	2,757860720e-01	5,515710115e-01
206	-7,653038000e-04	-1,869517611e-03	2,781755328e-01	5,563499928e-01
207	-1,054538000e-03	-2,205822384e-03	2,805356979e-01	5,610702634e-01
208	-1,360519200e-03	-2,553403843e-03	2,828655839e-01	5,657299161e-01
209	-1,683383000e-03	-2,911801683e-03	2,851640880e-01	5,703269839e-01
210	-2,022614600e-03	-3,280514618e-03	2,874303460e-01	5,748594403e-01
211	-2,377899500e-03	-3,659002949e-03	2,896633744e-01	5,793255568e-01
212	-2,748797700e-03	-4,046686925e-03	2,918621898e-01	5,837231875e-01
213	-3,134797500e-03	-4,442950245e-03	2,940258980e-01	5,880505443e-01
214	-3,535329200e-03	-4,847140983e-03	2,961534858e-01	5,923057795e-01
215	-3,949734800e-03	-5,258570891e-03	2,982441187e-01	5,964869261e-01
216	-4,377291000e-03	-5,676518660e-03	3,002967536e-01	6,005923152e-01
217	-4,817122000e-03	-6,100233644e-03	3,023106754e-01	6,046201587e-01
218	-5,268542300e-03	-6,528933067e-03	3,042849004e-01	6,085684896e-01
219	-5,730478300e-03	-6,961807609e-03	3,062185347e-01	6,124358177e-01
220	-6,202005100e-03	-7,398022339e-03	3,081108034e-01	6,162202954e-01
221	-6,681936000e-03	-7,836719044e-03	3,099608123e-01	6,199202538e-01
222	-7,167914500e-03	-8,277016692e-03	3,117676973e-01	6,235341430e-01
223	-7,662045500e-03	-8,718019351e-03	3,135308027e-01	6,270602942e-01
224	-8,160839200e-03	-9,158811532e-03	3,152491748e-01	6,304970384e-01
225	-8,663190500e-03	-9,598465636e-03	3,169221282e-01	6,338429451e-01
226	-9,169050700e-03	-1,003604382e-02	3,185488880e-01	6,370964646e-01
227	-9,675131500e-03	-1,047059800e-02	3,201287389e-01	6,402561665e-01
228	-1,018101800e-02	-1,090117730e-02	3,216609657e-01	6,433205605e-01
229	-1,068536400e-02	-1,132682897e-02	3,231448531e-01	6,462883353e-01
230	-1,118674000e-02	-1,174659748e-02	3,245797157e-01	6,491580606e-01
231	-1,168377500e-02	-1,215953380e-02	3,259649575e-01	6,519285440e-01
232	-1,217496400e-02	-1,256469358e-02	3,272998929e-01	6,545983553e-01
233	-1,265891600e-02	-1,296114177e-02	3,285838962e-01	6,571664810e-01
234	-1,313420500e-02	-1,334795821e-02	3,298164308e-01	6,596315503e-01
235	-1,359941000e-02	-1,372423489e-02	3,309969604e-01	6,619924903e-01
236	-1,405313100e-02	-1,408908330e-02	3,321248591e-01	6,642482877e-01
237	-1,449398400e-02	-1,444163360e-02	3,331996202e-01	6,663978696e-01
238	-1,492061500e-02	-1,478104480e-02	3,342207968e-01	6,684402227e-01

Index	32-Band Interpolation FIR		LFE Interpolation FIR	
	Perfect Reconstruction	Non-Perfect Reconstruction	64 x Interpolation	128 x Interpolation
239	-1,533170500e-02	-1,510649733e-02	3,351879120e-01	6,703743935e-01
240	-1,572581500e-02	-1,541720331e-02	3,361004293e-01	6,721994877e-01
241	-1,610200000e-02	-1,571240649e-02	3,369580209e-01	6,739146709e-01
242	-1,645893800e-02	-1,599138230e-02	3,377602994e-01	6,755192280e-01
243	-1,679548100e-02	-1,625344716e-02	3,385068178e-01	6,770122051e-01
244	-1,711054800e-02	-1,649795473e-02	3,391972482e-01	6,783930659e-01
245	-1,740312600e-02	-1,672429405e-02	3,398312926e-01	6,796611548e-01
246	-1,767225900e-02	-1,693190821e-02	3,404086530e-01	6,808158755e-01
247	-1,791707200e-02	-1,712027565e-02	3,409290314e-01	6,818566918e-01
248	-1,813675700e-02	-1,728892699e-02	3,413922191e-01	6,827830076e-01
249	-1,833061700e-02	-1,743743755e-02	3,417979777e-01	6,835945249e-01
250	-1,849797400e-02	-1,756543480e-02	3,421461284e-01	6,842908263e-01
251	-1,863830100e-02	-1,767260395e-02	3,424364924e-01	6,848715544e-01
252	-1,875108700e-02	-1,775865816e-02	3,426689506e-01	6,853365302e-01
253	-1,883604000e-02	-1,782339066e-02	3,428434134e-01	6,856853962e-01
254	-1,889315300e-02	-1,786663756e-02	3,429597318e-01	6,859180331e-01
255	-1,892151600e-02	-1,788828894e-02	3,430179358e-01	6,860344410e-01
256	1,892151600e-02	1,788828894e-02	3,430179358e-01	6,860344410e-01
257	1,889315300e-02	1,786663756e-02	3,429597318e-01	6,859180331e-01
258	1,883604000e-02	1,782339066e-02	3,428434134e-01	6,856853962e-01
259	1,875108700e-02	1,775865816e-02	3,426689506e-01	6,853365302e-01
260	1,863830100e-02	1,767260395e-02	3,424364924e-01	6,848715544e-01
261	1,849797400e-02	1,756543480e-02	3,421461284e-01	6,842908263e-01
262	1,833061700e-02	1,743743755e-02	3,417979777e-01	6,835945249e-01
263	1,813675700e-02	1,728892699e-02	3,413922191e-01	6,827830076e-01
264	1,791707200e-02	1,712027565e-02	3,409290314e-01	6,818566918e-01
265	1,767225900e-02	1,693190821e-02	3,404086530e-01	6,808158755e-01
266	1,740312600e-02	1,672429405e-02	3,398312926e-01	6,796611548e-01
267	1,711054800e-02	1,649795473e-02	3,391972482e-01	6,783930659e-01
268	1,679548100e-02	1,625344716e-02	3,385068178e-01	6,770122051e-01
269	1,645893800e-02	1,599138230e-02	3,377602994e-01	6,755192280e-01
270	1,610200000e-02	1,571240649e-02	3,369580209e-01	6,739146709e-01
271	1,572581500e-02	1,541720331e-02	3,361004293e-01	6,721994877e-01
272	1,533170500e-02	1,510649733e-02	3,351879120e-01	6,703743935e-01
273	1,492061500e-02	1,478104480e-02	3,342207968e-01	6,684402227e-01
274	1,449398400e-02	1,444163360e-02	3,331996202e-01	6,663978696e-01
275	1,405313100e-02	1,408908330e-02	3,321248591e-01	6,642482877e-01
276	1,359941000e-02	1,372423489e-02	3,309969604e-01	6,619924903e-01
277	1,313420500e-02	1,334795821e-02	3,298164308e-01	6,596315503e-01
278	1,265891600e-02	1,296114177e-02	3,285838962e-01	6,571664810e-01
279	1,217496400e-02	1,256469358e-02	3,272998929e-01	6,545983553e-01
280	1,168377500e-02	1,215953380e-02	3,259649575e-01	6,519285440e-01
281	1,118674000e-02	1,174659748e-02	3,245797157e-01	6,491580606e-01
282	1,068536400e-02	1,132682897e-02	3,231448531e-01	6,462883353e-01
283	1,018101800e-02	1,090117730e-02	3,216609657e-01	6,433205605e-01
284	9,675131500e-03	1,047059800e-02	3,201287389e-01	6,402561665e-01
285	9,169050700e-03	1,003604382e-02	3,185488880e-01	6,370964646e-01
286	8,663190500e-03	9,598465636e-03	3,169221282e-01	6,338429451e-01
287	8,160839200e-03	9,158811532e-03	3,152491748e-01	6,304970384e-01
288	7,662045500e-03	8,718019351e-03	3,135308027e-01	6,270602942e-01
289	7,167914500e-03	8,277016692e-03	3,117676973e-01	6,235341430e-01
290	6,681936000e-03	7,836719044e-03	3,099608123e-01	6,199202538e-01
291	6,202005100e-03	7,398022339e-03	3,081108034e-01	6,162202954e-01
292	5,730478300e-03	6,961807609e-03	3,062185347e-01	6,124358177e-01
293	5,268542300e-03	6,528933067e-03	3,042849004e-01	6,085684896e-01
294	4,817122000e-03	6,100233644e-03	3,023106754e-01	6,046201587e-01
295	4,377291000e-03	5,676518660e-03	3,002967536e-01	6,005923152e-01
296	3,949734800e-03	5,258570891e-03	2,982441187e-01	5,964869261e-01
297	3,535329200e-03	4,847140983e-03	2,961534858e-01	5,923057795e-01
298	3,134797500e-03	4,442950245e-03	2,940258980e-01	5,880505443e-01
299	2,748797700e-03	4,046686925e-03	2,918621898e-01	5,837231875e-01
300	2,377899500e-03	3,659002949e-03	2,896633744e-01	5,793255568e-01
301	2,022614600e-03	3,280514618e-03	2,874303460e-01	5,748594403e-01

Index	32-Band Interpolation FIR		LFE Interpolation FIR	
	Perfect Reconstruction	Non-Perfect Reconstruction	64 x Interpolation	128 x Interpolation
302	1,683383000e-03	2,911801683e-03	2,851640880e-01	5,703269839e-01
303	1,360519200e-03	2,553403843e-03	2,828655839e-01	5,657299161e-01
304	1,054538000e-03	2,205822384e-03	2,805356979e-01	5,610702634e-01
305	7,653038000e-04	1,869517611e-03	2,781755328e-01	5,563499928e-01
306	4,930996000e-04	1,544908970e-03	2,757860720e-01	5,515710115e-01
307	2,380696500e-04	1,232374110e-03	2,733682692e-01	5,467353463e-01
308	2,272228400e-07	9,322494152e-04	2,709231377e-01	5,418450832e-01
309	-2,204804700e-04	6,448282511e-04	2,684516609e-01	5,369022489e-01
310	-4,241331700e-04	3,703625989e-04	2,659549415e-01	5,319088101e-01
311	-6,109076600e-04	1,090620208e-04	2,634339035e-01	5,268667936e-01
312	-7,810380900e-04	-1,389056415e-04	2,608896792e-01	5,217782855e-01
313	-9,349224800e-04	-3,734139318e-04	2,583232224e-01	5,166453719e-01
314	-1,072608600e-03	-5,943773431e-04	2,557355762e-01	5,114701390e-01
315	-1,194737700e-03	-8,017504588e-04	2,531278133e-01	5,062545538e-01
316	-1,301623400e-03	-9,955273708e-04	2,505008876e-01	5,010007620e-01
317	-1,394017000e-03	-1,175740734e-03	2,478559017e-01	4,957108200e-01
318	-1,478566700e-03	-1,342460280e-03	2,451938838e-01	4,903867543e-01
319	-1,541196600e-03	-1,495792647e-03	2,425158769e-01	4,850307405e-01
320	1,592423900e-03	1,635878929e-03	2,398228943e-01	4,796448052e-01
321	1,632849400e-03	1,762894331e-03	2,371159792e-01	4,742309451e-01
322	1,654649800e-03	1,877046190e-03	2,343961596e-01	4,687913656e-01
323	1,670887400e-03	1,978572691e-03	2,316644788e-01	4,633280039e-01
324	1,676540900e-03	2,067740774e-03	2,289219648e-01	4,578429461e-01
325	1,672691700e-03	2,144844970e-03	2,261696160e-01	4,523383081e-01
326	1,659751400e-03	2,210205887e-03	2,234084606e-01	4,468160272e-01
327	1,638393400e-03	2,264167881e-03	2,206395119e-01	4,412781000e-01
328	1,609496400e-03	2,307097195e-03	2,178637981e-01	4,357266724e-01
329	1,573715600e-03	2,339380793e-03	2,150822729e-01	4,301636219e-01
330	1,531686400e-03	2,361423569e-03	2,122959495e-01	4,245910645e-01
331	1,484159500e-03	2,373647178e-03	2,095058411e-01	4,190107882e-01
332	1,431717500e-03	2,376487479e-03	2,067128718e-01	4,134248793e-01
333	1,375058300e-03	2,370394068e-03	2,039180547e-01	4,078352153e-01
334	1,314813200e-03	2,355825622e-03	2,011223286e-01	4,022437930e-01
335	1,251459700e-03	2,333251061e-03	1,983266175e-01	3,966524303e-01
336	1,182107000e-03	2,303145360e-03	1,955319196e-01	3,910630047e-01
337	1,114606900e-03	2,265989315e-03	1,927391142e-01	3,854774535e-01
338	1,045985500e-03	2,222266514e-03	1,899491698e-01	3,798975349e-01
339	9,767158300e-04	2,172462177e-03	1,871629506e-01	3,743250966e-01
340	9,072555100e-04	2,117061289e-03	1,843813360e-01	3,687619269e-01
341	8,380918900e-04	2,056547208e-03	1,816052496e-01	3,632097244e-01
342	7,695597500e-04	1,991399564e-03	1,788355410e-01	3,576703668e-01
343	7,021067600e-04	1,922092517e-03	1,760730892e-01	3,521454632e-01
344	6,360244700e-04	1,849094522e-03	1,733186990e-01	3,466366828e-01
345	5,717321000e-04	1,772865304e-03	1,705732346e-01	3,411457539e-01
346	5,091327500e-04	1,693855622e-03	1,678375006e-01	3,356742859e-01
347	4,488403900e-04	1,612505526e-03	1,651122719e-01	3,302238286e-01
348	3,909221300e-04	1,529243193e-03	1,623983532e-01	3,247960210e-01
349	3,353960600e-04	1,444484224e-03	1,596965194e-01	3,193923831e-01
350	2,678985000e-04	1,358630019e-03	1,570075154e-01	3,140144050e-01
351	2,231129500e-04	1,272067428e-03	1,543320864e-01	3,086635172e-01
352	1,772621900e-04	1,185167348e-03	1,516709626e-01	3,033412695e-01
353	1,311540500e-04	1,098284614e-03	1,490248144e-01	2,980490029e-01
354	1,052143400e-04	1,011756598e-03	1,463943720e-01	2,927881181e-01
355	6,898332300e-05	9,259034996e-04	1,437802613e-01	2,875599265e-01
356	3,623958500e-05	8,410271257e-04	1,411831975e-01	2,823657692e-01
357	6,512868000e-06	7,574109477e-04	1,386037618e-01	2,772069275e-01
358	-2,010055900e-05	6,753197522e-04	1,360425949e-01	2,720846236e-01
359	-4,214289700e-05	5,949990009e-04	1,335003078e-01	2,670000792e-01
360	-6,157321800e-05	5,166754709e-04	1,309774816e-01	2,619544268e-01
361	-7,793692700e-05	4,405563814e-04	1,284746826e-01	2,569487989e-01
362	-9,130172200e-05	3,668301215e-04	1,259924471e-01	2,519843280e-01
363	-1,018237000e-04	2,956659591e-04	1,235313043e-01	2,470620573e-01
364	-1,097435100e-04	2,272142592e-04	1,210917681e-01	2,421830446e-01

Index	32-Band Interpolation FIR		LFE Interpolation FIR	
	Perfect Reconstruction	Non-Perfect Reconstruction	64 x Interpolation	128 x Interpolation
365	-1,152534400e-04	1,616069785e-04	1,186743453e-01	2,373482138e-01
366	-1,185602000e-04	9,895755647e-05	1,162794977e-01	2,325585187e-01
367	-1,200453700e-04	3,936182839e-05	1,139076948e-01	2,278149277e-01
368	-1,237377900e-04	-1,710198012e-05	1,115593687e-01	2,231182903e-01
369	-1,226936800e-04	-7,037253090e-05	1,092349365e-01	2,184694260e-01
370	-1,202018700e-04	-1,204049113e-04	1,069347933e-01	2,138691545e-01
371	-1,165901700e-04	-1,671699720e-04	1,046593264e-01	2,093182206e-01
372	-1,119841200e-04	-2,106537577e-04	1,024089083e-01	2,048173845e-01
373	-1,064814700e-04	-2,508567995e-04	1,001838669e-01	2,003673166e-01
374	-1,002681400e-04	-2,877934603e-04	9,798453748e-02	1,959686577e-01
375	-9,341758200e-05	-3,214911267e-04	9,581124038e-02	1,916220933e-01
376	-8,597821200e-05	-3,519894381e-04	9,366425127e-02	1,873281151e-01
377	-7,984114900e-05	-3,793396754e-04	9,154383838e-02	1,830873191e-01
378	-7,052899300e-05	-4,036037717e-04	8,945026249e-02	1,789001823e-01
379	-6,144976200e-05	-4,248536134e-04	8,738376945e-02	1,747671962e-01
380	-5,311715600e-05	-4,431701091e-04	8,534456789e-02	1,706887931e-01
381	-4,511232400e-05	-4,586426076e-04	8,333285898e-02	1,666653752e-01
382	-1,774116500e-05	-4,713678791e-04	8,134882897e-02	1,626973301e-01
383	-1,994364800e-05	-4,814492422e-04	7,939263433e-02	1,587849557e-01
384	1,610369300e-05	4,889960401e-04	7,746443897e-02	1,549285650e-01
385	1,518084500e-05	4,941228544e-04	7,556436211e-02	1,511284113e-01
386	1,614022200e-07	4,969481961e-04	7,369252294e-02	1,473847479e-01
387	-7,179248900e-06	4,975944757e-04	7,184901088e-02	1,436977387e-01
388	-1,390508100e-05	4,961824161e-04	7,003392279e-02	1,400675476e-01
389	-1,979628700e-05	4,928477574e-04	6,824731827e-02	1,364943385e-01
390	-2,497457200e-05	4,877146275e-04	6,648923457e-02	1,329781860e-01
391	-2,291622100e-05	4,809123348e-04	6,475970894e-02	1,295191795e-01
392	-2,354812700e-05	4,725702747e-04	6,305878609e-02	1,261173040e-01
393	-2,295038200e-05	4,628172028e-04	6,138643622e-02	1,227726117e-01
394	-2,131957100e-05	4,517810594e-04	5,974265561e-02	1,194850579e-01
395	-1,881671400e-05	4,395879805e-04	5,812742189e-02	1,162546203e-01
396	-1,561346600e-05	4,263620067e-04	5,654069409e-02	1,130811572e-01
397	-1,191309700e-05	4,122247046e-04	5,498242006e-02	1,099646092e-01
398	-7,916183300e-06	3,972945851e-04	5,345252529e-02	1,069048345e-01
399	-3,803557300e-06	3,816867538e-04	5,195093155e-02	1,039016470e-01
400	-5,236303900e-05	3,655125911e-04	5,047753453e-02	1,009548605e-01
401	-5,081695700e-05	3,488793736e-04	4,903224111e-02	9,806428105e-02
402	-4,846834800e-05	3,318900708e-04	4,761491716e-02	9,522963315e-02
403	-4,534151200e-05	3,146430245e-04	4,622544348e-02	9,245070815e-02
404	-4,149260300e-05	2,972317743e-04	4,486365616e-02	8,972713351e-02
405	-3,698112500e-05	2,797449124e-04	4,352942482e-02	8,705867827e-02
406	-3,196094400e-05	2,622658503e-04	4,222255200e-02	8,444493264e-02
407	-2,647159500e-05	2,448728774e-04	4,094288871e-02	8,188561350e-02
408	-2,089583800e-05	2,276388550e-04	3,969023004e-02	7,938029617e-02
409	-1,073666400e-05	2,106313768e-04	3,846437484e-02	7,692859322e-02
410	-7,240269200e-06	1,939126523e-04	3,726511076e-02	7,453006506e-02
411	-4,157326000e-06	1,775395358e-04	3,609224036e-02	7,218432426e-02
412	-1,809597500e-06	1,615635992e-04	3,494550660e-02	6,989086419e-02
413	-7,763173700e-08	1,460311323e-04	3,382468969e-02	6,764923781e-02
414	1,846174800e-06	1,309833024e-04	3,272953629e-02	6,545893103e-02
415	1,466421500e-06	1,164562127e-04	3,165979683e-02	6,331945211e-02
416	9,974569000e-07	1,024810626e-04	3,061520495e-02	6,123027951e-02
417	5,671807300e-07	8,908427117e-05	2,959549613e-02	5,919086933e-02
418	6,805568100e-06	7,628766616e-05	2,860039286e-02	5,720067024e-02
419	1,128458200e-05	6,410864444e-05	2,762960829e-02	5,525910854e-02
420	1,568432200e-05	5,256036457e-05	2,668286115e-02	5,336561054e-02
421	1,771374800e-05	4,165195787e-05	2,575986087e-02	5,151961371e-02
422	1,870056200e-05	3,138873581e-05	2,486028522e-02	4,972046614e-02
423	2,571454500e-05	2,177240640e-05	2,398385666e-02	4,796761274e-02
424	2,861654300e-05	1,280130618e-05	2,313023806e-02	4,626038298e-02
425	3,109003600e-05	4,470633485e-06	2,229913883e-02	4,459818453e-02
426	3,280414400e-05	-3,227306706e-06	2,149021253e-02	4,298033938e-02
427	3,382472000e-05	-1,030297699e-05	2,070316114e-02	4,140623659e-02

Index	32-Band Interpolation FIR		LFE Interpolation FIR	
	Perfect Reconstruction	Non-Perfect Reconstruction	64 x Interpolation	128 x Interpolation
428	3,411568400e-05	-1,676913780e-05	1,993762329e-02	3,987516090e-02
429	3,367812600e-05	-2,264085742e-05	1,919330470e-02	3,838652745e-02
430	3,254459900e-05	-2,793515523e-05	1,846982725e-02	3,693958372e-02
431	3,076839000e-05	-3,267079956e-05	1,776690222e-02	3,553372994e-02
432	-1,987389300e-05	-3,686808850e-05	1,708412915e-02	3,416819125e-02
433	-1,925789500e-05	-4,054862075e-05	1,642123051e-02	3,284239396e-02
434	-1,819741000e-05	-4,373511547e-05	1,577781141e-02	3,155555204e-02
435	-1,676702500e-05	-4,645117951e-05	1,515355054e-02	3,030703776e-02
436	-1,505747500e-05	-4,872112549e-05	1,454808749e-02	2,909611352e-02
437	-1,317522400e-05	-5,056979353e-05	1,396108977e-02	2,792212367e-02
438	-1,121856000e-05	-5,202236207e-05	1,339218579e-02	2,678431384e-02
439	-9,321632700e-06	-5,310418419e-05	1,284105983e-02	2,568206564e-02
440	-7,476824700e-06	-5,384063843e-05	1,230732165e-02	2,461459488e-02
441	-3,262621300e-06	-5,425697600e-05	1,179065090e-02	2,358125709e-02
442	-3,742137100e-06	-5,437819709e-05	1,129068248e-02	2,258131653e-02
443	-3,694976800e-06	-5,422891263e-05	1,080708485e-02	2,161412500e-02
444	-2,480143600e-06	-5,383323878e-05	1,033949479e-02	2,067894675e-02
445	-1,236415900e-06	-5,321469871e-05	9,887560271e-03	1,977507770e-02
446	-6,479789100e-08	-5,239612074e-05	9,450953454e-03	1,890186779e-02
447	-8,454083600e-08	-5,139957648e-05	9,029330686e-03	1,805862412e-02
448	3,772031200e-08	5,024627535e-05	8,622321300e-03	1,724460535e-02
449	3,227406600e-08	4,895655002e-05	8,229630999e-03	1,645922661e-02
450	-3,384827600e-08	4,754976908e-05	7,850865833e-03	1,570170000e-02
451	-1,510238900e-08	4,604430433e-05	7,485736627e-03	1,497144438e-02
452	-5,699348500e-08	4,445751256e-05	7,133882027e-03	1,426773332e-02
453	-3,073465500e-07	4,280570283e-05	6,794991903e-03	1,358995494e-02
454	-6,650809100e-07	4,110412556e-05	6,468691397e-03	1,293735672e-02
455	-3,139397100e-06	3,936696885e-05	6,154712290e-03	1,230939943e-02
456	-4,187209700e-06	3,760734762e-05	5,852684379e-03	1,170534454e-02
457	-5,044609200e-06	3,583733633e-05	5,562345497e-03	1,112466771e-02
458	-5,908209500e-06	3,406793985e-05	5,283284001e-03	1,056654565e-02
459	-6,629955000e-06	3,230916263e-05	5,015311297e-03	1,003060210e-02
460	-7,179758900e-06	3,056998685e-05	4,758012015e-03	9,516004470e-03
461	-7,533601500e-06	2,885844333e-05	4,511159845e-03	9,022301060e-03
462	-7,678809700e-06	2,718161704e-05	4,274417181e-03	8,548815730e-03
463	-7,614387900e-06	2,554569073e-05	4,047499038e-03	8,094980380e-03
464	-6,046428700e-06	2,395598858e-05	3,830091329e-03	7,660165890e-03
465	-5,696789700e-06	2,241701623e-05	3,621967277e-03	7,243919190e-03
466	-5,210775600e-06	2,093250441e-05	3,422776936e-03	6,845539900e-03
467	-4,617880200e-06	1,950545993e-05	3,232272575e-03	6,464532110e-03
468	-3,953604500e-06	1,813820381e-05	3,050152911e-03	6,100293250e-03
469	-3,253336000e-06	1,683242772e-05	2,876190469e-03	5,752369300e-03
470	-2,555774300e-06	1,558924305e-05	2,710093278e-03	5,420174920e-03
471	-1,904890000e-06	1,440921824e-05	2,551567042e-03	5,103122910e-03
472	-1,306777300e-06	1,329243969e-05	2,400433412e-03	4,800856580e-03
473	-2,448299500e-07	1,223855270e-05	2,256359672e-03	4,512710030e-03
474	-1,007547900e-07	1,124680875e-05	2,119151875e-03	4,238294900e-03
475	-1,643020800e-08	1,031611009e-05	1,988604199e-03	3,977200480e-03
476	-3,715261200e-09	9,445050637e-06	1,864377526e-03	3,728747140e-03
477	-6,820855300e-09	8,631964192e-06	1,746327500e-03	3,492647550e-03
478	3,377455500e-09	7,874960829e-06	1,634211512e-03	3,268416510e-03
479	2,668096400e-09	7,171964626e-06	1,527829794e-03	3,055653300e-03
480	1,611726200e-09	6,520755960e-06	1,426893868e-03	2,853781920e-03
481	6,821591800e-10	5,918994248e-06	1,331258914e-03	2,662512240e-03
482	-4,147967800e-08	5,364252502e-06	1,240676851e-03	2,481348810e-03
483	-1,028149000e-07	4,854050530e-06	1,155023579e-03	2,310042500e-03
484	-3,167104300e-07	4,385879038e-06	1,073930296e-03	2,147856400e-03
485	-8,454480100e-07	3,957220997e-06	9,974770946e-04	1,994950230e-03
486	-1,560941800e-06	3,565570978e-06	9,251192096e-04	1,850234690e-03
487	-8,920065800e-07	3,208459020e-06	8,570110658e-04	1,714018640e-03
488	-9,822894700e-07	2,883470643e-06	7,928516716e-04	1,585700080e-03
489	-1,124542400e-06	2,588257530e-06	7,324148901e-04	1,464826870e-03
490	-1,144180200e-06	2,320550948e-06	6,755515351e-04	1,351100280e-03

Index	32-Band Interpolation FIR		LFE Interpolation FIR	
	Perfect Reconstruction	Non-Perfect Reconstruction	64 x Interpolation	128 x Interpolation
491	-1,119829700e-06	2,078171747e-06	6,222130032e-04	1,244423330e-03
492	-1,052683900e-06	1,859034001e-06	5,721592461e-04	1,144316160e-03
493	-9,544782800e-07	1,661159786e-06	5,252459669e-04	1,050489840e-03
494	-8,348606100e-07	1,482681114e-06	4,812776169e-04	9,625531400e-04
495	-7,034745600e-07	1,321840614e-06	4,401875485e-04	8,803732300e-04
496	5,586146700e-07	1,176999604e-06	4,018281470e-04	8,036546600e-04
497	6,193701600e-07	1,046637067e-06	3,658991191e-04	7,317967800e-04
498	6,601852300e-07	9,293416952e-07	3,328395542e-04	6,656776500e-04
499	6,770656500e-07	8,238164355e-07	3,013863170e-04	6,027714100e-04
500	6,677818900e-07	7,288739425e-07	2,726115927e-04	5,452220800e-04
501	6,300390500e-07	6,434325428e-07	2,460231190e-04	4,920452500e-04
502	5,697322600e-07	5,665120852e-07	2,211847313e-04	4,423685900e-04
503	4,468735700e-07	4,972276315e-07	1,981738606e-04	3,963469100e-04
504	3,712986200e-07	4,347855338e-07	1,769922383e-04	3,539837500e-04
505	5,746147600e-07	3,784752209e-07	1,575958013e-04	3,151909600e-04
506	2,771560000e-07	3,276689142e-07	1,397485757e-04	2,794966000e-04
507	9,130198800e-08	2,818143514e-07	1,233371440e-04	2,466738200e-04
508	2,529296600e-08	2,404238444e-07	1,082170274e-04	2,164336300e-04
509	8,358679600e-09	2,030677564e-07	9,439323912e-05	1,887860900e-04
510	-7,138742100e-11	1,693738625e-07	8,179365250e-05	1,635869100e-04
511	-1,140033200e-10	1,390191784e-07	2,658434387e-04	5,316857100e-04

D.9 1 024 tap FIR for X96 Synthesis QMF

Coef #	Coefficient Value
1	-7,1279389866041690e-8
2	-7,0950903150874990e-8
3	-7,9525034321375090e-8
4	-8,9326871281374790e-8
5	-9,7483190494874640e-8
6	-1,0586659510721950e-7
7	-1,1527363411865760e-7
8	-1,2523109645615350e-7
9	-1,3555636137971380e-7
10	-1,4640931821191640e-7
11	-1,5787424998123710e-7
12	-1,6994592926571220e-7
13	-1,8264184311018150e-7
14	-1,9600594587757850e-7
15	-2,1007899706006800e-7
16	-2,2489759047553620e-7
17	-2,4050245877923400e-7
18	-2,5693874615777660e-7
19	-2,7425370308549580e-7
20	-2,9249662541757600e-7
21	-3,1171981968268900e-7
22	-3,3197864040473180e-7
23	-3,5333134588843600e-7
24	-3,7583910465412180e-7
25	-3,9956628121669190e-7
26	-4,2458045296159330e-7
27	-4,5095249563698380e-7
28	-4,7875659112606760e-7
29	-5,0807031294631050e-7
30	-5,3897464223335550e-7
31	-5,7155423388793000e-7
32	-6,0589741192019800e-7
33	-6,4209580087853050e-7
34	-6,8024432849991400e-7
35	-7,2044165259013450e-7
36	-7,6279018172077200e-7
37	-8,0739610368280600e-7
38	-8,5436937020293650e-7
39	-9,0382333224895500e-7
40	-9,5587472540498280e-7
41	-1,0106437128213760e-6
42	-1,0682538906146330e-6
43	-1,1288323637155430e-6
44	-1,1925097184646100e-6
45	-1,2594193438633260e-6
46	-1,3296974052300520e-6
47	-1,4034829433780260e-6
48	-1,4809178634979940e-6
49	-1,5621465913980060e-6
50	-1,6473160377545360e-6
51	-1,7365751553971340e-6
52	-1,8300749104526120e-6
53	-1,9279680902322050e-6
54	-2,0304092660990760e-6
55	-2,1375541686662320e-6
56	-2,2495596253776460e-6
57	-2,3665828120950560e-6
58	-2,4887811970856540e-6
59	-2,6163120571644820e-6
60	-2,7493324266391140e-6
61	-2,8879984579501380e-6
62	-3,0324653639203880e-6
63	-3,1828867865974640e-6
64	-3,3394147240855820e-6
65	-3,5021985520037540e-6
66	-3,6713849262294680e-6
67	-3,8471166218015830e-6
68	-4,0295324657699800e-6
69	-4,2187670209852250e-6
70	-4,4149505043312320e-6
71	-4,6182073040543900e-6
72	-4,8286558658894680e-6
73	-5,0464076705003400e-6
74	-5,2715671455405850e-6
75	-5,5042307535458100e-6
76	-5,7444868785028750e-6
77	-5,9924142424804800e-6
78	-6,2480817863072800e-6
79	-6,5115476278604000e-6
80	-6,7828589565804950e-6
81	-7,0620507545154100e-6
82	-7,3491456697144900e-6
83	-7,6441525098592000e-6
84	-7,9470661364365490e-6
85	-8,2578666429227760e-6
86	-8,5765192154768100e-6
87	-8,9029718900122000e-6
88	-9,2371554195468400e-6
89	-9,5789825988253100e-6
90	-9,9283481667264540e-6
91	-1,0285127334608450e-5
92	-1,0649175634328590e-5
93	-1,1020327046334650e-5
94	-1,1398393880881060e-5
95	-1,1783166036693560e-5
96	-1,2174410889256530e-5
97	-1,2571871574469220e-5
98	-1,2975266858030310e-5
99	-1,3384289978242060e-5
100	-1,3798608540965960e-5
101	-1,4218763365843780e-5
102	-1,4644668383496450e-5
103	-1,5076609527679710e-5
104	-1,5501244597782600e-5
105	-1,5936101341640500e-5
106	-1,6373677426466900e-5
107	-1,6813441716539560e-5
108	-1,7254834143068570e-5
109	-1,7697261563607500e-5
110	-1,8140097714618200e-5
111	-1,8582686308483550e-5
112	-1,9024340970099240e-5
113	-1,9464340744880170e-5
114	-1,9901930030141080e-5
115	-2,0336321557464510e-5
116	-2,0766696417773940e-5
117	-2,1192201630574280e-5
118	-2,1611950084231910e-5
119	-2,2025021652894630e-5
120	-2,2430463239517170e-5
121	-2,2827288605503470e-5
122	-2,3214478371918170e-5
123	-2,3590980216497750e-5
124	-2,3955708939763280e-5
125	-2,4307547722489910e-5
126	-2,4645348200903720e-5
127	-2,4967930863542620e-5
128	-2,5274085082324960e-5
129	2,5562569399199020e-5
130	2,5832111724562040e-5
131	2,6081413415833200e-5
132	2,6309149390520640e-5
133	2,6513966535124460e-5
134	2,6694483850464800e-5
135	2,6849297239997120e-5
136	2,6976979753245170e-5
137	2,7076082153012070e-5
138	2,7145133103663890e-5
139	2,7182642746157720e-5
140	2,7187102980536760e-5
141	2,7156990105794550e-5
142	2,7090765064735940e-5
143	2,6986876191054700e-5
144	2,6843759539035120e-5
145	2,6659843389908530e-5
146	2,6433548592482960e-5
147	2,6163291550639320e-5
148	2,5847484577439320e-5
149	2,5484540698002960e-5
150	2,5072874052447260e-5
151	2,4610903957661350e-5
152	2,4097055300584120e-5
153	2,3529763128539150e-5
154	2,2907473117403740e-5
155	2,2228647280994560e-5
156	2,1491764461832140e-5
157	2,0695325418576370e-5
158	1,9837853265502040e-5
159	1,8917898053528080e-5
160	1,7934037335534700e-5
160	1,7934037335534700e-5
161	1,6884884022311310e-5
162	1,5769086921577900e-5
163	1,4585334740507920e-5
164	1,3332356590263080e-5
165	1,2008929083235120e-5
166	1,0613876973541650e-5
167	9,1460801506922300e-6
168	7,6044740757265500e-6
169	5,9880523825897000e-6
170	4,2958677847708840e-6
171	2,5270494199802070e-6
172	6,8080315386959900e-7
173	-1,2435990827839660e-6
174	-3,2468187066945930e-6
175	-5,3294268968052800e-6
176	-7,4919036559427900e-6
177	-9,7346407524871100e-6
178	-1,2057941408635810e-5
179	-1,4462010500550970e-5
180	-1,6946953848107710e-5
181	-1,9512772389215240e-5
182	-2,2159361597292710e-5
183	-2,4886504491416160e-5
184	-2,7693871041862060e-5
185	-3,0581012081916960e-5
186	-3,3547358765259040e-5
187	-3,6592216716540970e-5
188	-3,9714765485539260e-5
189	-4,2914052387856640e-5
190	-4,6188991961724560e-5
191	-4,9538360176954340e-5

Coef #	Coefficient Value
192	-5,2960794000140600e-5
193	-5,6454787620185350e-5
194	-6,0018691972791050e-5
195	-6,3650708053788550e-5
196	-6,7348886525882700e-5
197	-7,1111125753722860e-5
198	-7,4935171411178040e-5
199	-7,8818609806320000e-5
200	-8,2758867628623800e-5
201	-8,6753213062403400e-5
202	-9,0798755482139240e-5
203	-9,4892437636254000e-5
204	-9,9031035415475500e-5
205	-1,0321116057304920e-4
206	-1,0742926070674010e-4
207	-1,1168161614264270e-4
208	-1,1596433966772030e-4
209	-1,2027337376184880e-4
210	-1,2460449089034750e-4
211	-1,2895330270761850e-4
212	-1,3331525983885840e-4
213	-1,3768563787408340e-4
214	-1,4205953760578120e-4
215	-1,4643190426323540e-4
216	-1,5079752791692290e-4
217	-1,5515103314094370e-4
218	-1,5948687909058970e-4
219	-1,6379937154642740e-4
220	-1,6808266356446820e-4
221	-1,7233075773584680e-4
222	-1,7653750646560020e-4
223	-1,8069661584612290e-4
224	-1,8480164638953740e-4
225	-1,8884602526638050e-4
226	-1,9282304696531290e-4
227	-1,9672587556804270e-4
228	-2,0054754547108610e-4
229	-2,0428097498957360e-4
230	-2,0791896735659270e-4
231	-2,1145421910491280e-4
232	-2,1487932080488700e-4
233	-2,1818676491655660e-4
234	-2,2136894706917190e-4
235	-2,2441818635854840e-4
236	-2,2732672676686290e-4
237	-2,3008674810081220e-4
238	-2,3269036653525150e-4
239	-2,3512963563507800e-4
240	-2,3739654799291240e-4
241	-2,3948307023621970e-4
242	-2,4138114593372380e-4
243	-2,4308272452941820e-4
244	-2,4457975717687980e-4
245	-2,4586407616017000e-4
246	-2,4692741569938220e-4
247	-2,4776199025177680e-4
248	-2,4836046092637280e-4
249	-2,4871461830381660e-4
250	-2,4881540457247040e-4
251	-2,4865471770248810e-4
252	-2,4822543333608940e-4
253	-2,4752008219951220e-4
254	-2,4653081794176570e-4
255	-2,4525003217524440e-4

Coef #	Coefficient Value
256	-2,4367037437341020e-4
257	2,4178457457919800e-4
258	2,3958543985834340e-4
259	2,3706595356421380e-4
260	2,3421928008696620e-4
261	2,3103877007019660e-4
262	2,2751796132972940e-4
263	2,2365059385031200e-4
264	2,1943061159567880e-4
265	2,1485218733324290e-4
266	2,0990972538755550e-4
267	2,0459789739171640e-4
268	1,9891164424316180e-4
269	1,9284618337957900e-4
270	1,8639701070854740e-4
271	1,7955993578371750e-4
272	1,7233108395155490e-4
273	1,6470690840351980e-4
274	1,5668419239382670e-4
275	1,4826008597798460e-4
276	1,3943210818407590e-4
277	1,3019815804235450e-4
278	1,2055651618705740e-4
279	1,1050586906665980e-4
280	1,0004531116378850e-4
281	8,9174369623037550e-5
282	7,7893006105444910e-5
283	6,6201632959158800e-5
284	5,4101115008963200e-5
285	4,1592792755171040e-5
286	2,8678484003389880e-5
287	1,5360496520469550e-5
288	1,6416297019304470e-6
289	-1,2474801400564880e-5
290	-2,6984964757674160e-5
291	-4,1884506316022560e-5
292	-5,7168548169515050e-5
293	-7,2831653374563650e-5
294	-8,8867825335203200e-5
295	-1,0527052946041040e-4
296	-1,2203269185287530e-4
297	-1,3914664880465400e-4
298	-1,5660414556657680e-4
299	-1,7439635977956720e-4
300	-1,9251390142613000e-4
301	-2,1094678833176480e-4
302	-2,2968444457655940e-4
303	-2,4871569004884240e-4
304	-2,6802874127778880e-4
305	-2,8761124019708440e-4
306	-3,0745025297166030e-4
307	-3,2753221542219500e-4
308	-3,4784293347385130e-4
309	-3,6836764760397900e-4
310	-3,8909103335715320e-4
311	-4,0999714822829530e-4
312	-4,3106943097409840e-4
313	-4,5229073958127750e-4
314	-4,7364335350426610e-4
315	-4,9510898492628230e-4
316	-5,1666877822680600e-4
317	-5,3830328700499900e-4
318	-5,5999247625413450e-4
319	-5,8171579317559550e-4

Coef #	Coefficient Value
320	-6,0345216810040500e-4
321	-6,2517996394886800e-4
322	-6,4687697763330650e-4
323	-6,6852052145861900e-4
324	-6,9008742526480550e-4
325	-7,1155400216196750e-4
326	-7,3289605035434600e-4
327	-7,5408892753413600e-4
328	-7,7510755325530950e-4
329	-7,9592638676398950e-4
330	-8,1651942936660300e-4
331	-8,3686029869896850e-4
332	-8,5692223230171550e-4
333	-8,7667809201044900e-4
334	-8,9610036547340950e-4
335	-9,1516119514986000e-4
336	-9,3383238324995840e-4
337	-9,5208547141354700e-4
338	-9,6989174214394400e-4
339	-9,8722217067251400e-4
340	-1,0040474297445790e-3
341	-1,0203380430448350e-3
342	-1,0360643900964910e-3
343	-1,0511966606168980e-3
344	-1,0657048556038330e-3
345	-1,0795588568782010e-3
346	-1,0927284332850520e-3
347	-1,1051833075734240e-3
348	-1,1168931605293980e-3
349	-1,1278276550234160e-3
350	-1,1379564405661050e-3
351	-1,1472492294891640e-3
352	-1,1556758000499310e-3
353	-1,1632059885511800e-3
354	-1,1698096960154200e-3
355	-1,1754570428998680e-3
356	-1,1801183720212010e-3
357	-1,1837641581852870e-3
358	-1,1863650153060240e-3
359	-1,1878919433902330e-3
360	-1,1883163316843900e-3
361	-1,1876097809178440e-3
362	-1,1857441072376680e-3
363	-1,1826916065321440e-3
364	-1,1784250628415020e-3
365	-1,1729176690707690e-3
366	-1,1661430294353880e-3
367	-1,1580752925519680e-3
368	-1,1486891568700730e-3
369	-1,1379598570807350e-3
370	-1,1258631690530700e-3
371	-1,1123755320313420e-3
372	-1,0974740543942100e-3
373	-1,0811365181610850e-3
374	-1,0633413819325920e-3
375	-1,0440678410743190e-3
376	-1,0232958338441730e-3
377	-1,0010061160103870e-3
378	-9,7718026613602800e-4
379	-9,5180072722007060e-4
380	-9,2485080945546600e-4
381	-8,9631470926558100e-4
382	-8,6617751390929650e-4
383	-8,3442528375666700e-4

Coef #	Coefficient Value
384	-8,0104505655578600e-4
385	7,6602485902806300e-4
386	7,2935371157849700e-4
387	6,9102172032639350e-4
388	6,5102007948960100e-4
389	6,0934103180698750e-4
390	5,6597787244591100e-4
391	5,2092507454409450e-4
392	4,7417829303612970e-4
393	4,2573432329476060e-4
394	3,7559110239316400e-4
395	3,2374777823389440e-4
396	2,7020471361387040e-4
397	2,1496350652513040e-4
398	1,5802699163104190e-4
399	9,9399252442146460e-5
400	3,9085623279149610e-5
401	-2,2907279717153710e-5
402	-8,6571556658577800e-5
403	-1,5189801251884410e-4
404	-2,1887615668780720e-4
405	-2,8749420455512610e-4
406	-3,5773907519488100e-4
407	-4,2959633884346400e-4
408	-5,0305021853266900e-4
409	-5,7808367853976800e-4
410	-6,5467842279709950e-4
411	-7,3281477145293300e-4
412	-8,1247166091487090e-4
413	-8,9362676824381250e-4
414	-9,7625651350435400e-4
415	-1,0603359870796300e-3
416	-1,1458389497708370e-3
417	-1,2327379076720450e-3
418	-1,3210041125919850e-3
419	-1,4106074964587530e-3
420	-1,5015166759080980e-3
421	-1,5936991188210680e-3
422	-1,6871211446584360e-3
423	-1,7817477652804130e-3
424	-1,8775426862784720e-3
425	-1,9744684954557180e-3
426	-2,0724866717198650e-3
427	-2,1715575922482770e-3
428	-2,2716405301701940e-3
429	-2,3726935964105480e-3
430	-2,4746737418493600e-3
431	-2,5775368629845300e-3
432	-2,6812378156433060e-3
433	-2,7857306109716460e-3
434	-2,8909684129992950e-3
435	-2,9969032890820140e-3
436	-3,1034862132032660e-3
437	-3,2106673881690480e-3
438	-3,3183962566563890e-3
439	-3,4266214220855450e-3
440	-3,5352906520539890e-3
441	-3,6443510330122110e-3
442	-3,7537489709635930e-3
443	-3,8634300521859240e-3
444	-3,9733390588103910e-3
445	-4,0834204508378470e-3
446	-4,1936183686073460e-3
447	-4,3038762050876900e-3
448	-4,4141366059086620e-3
449	-4,5243418977567100e-3
450	-4,6344341075783170e-3
451	-4,7443549566589130e-3
452	-4,8540458565358850e-3
453	-4,9634478250060780e-3
454	-5,0725014978718950e-3
455	-5,1811474713480250e-3
456	-5,2893263173827950e-3
457	-5,3969785783205650e-3
458	-5,5040447592375600e-3
459	-5,6104651646665850e-3
460	-5,7161799140263450e-3
461	-5,8211294443412250e-3
462	-5,9252545216101900e-3
463	-6,0284959882028950e-3
464	-6,1307947602360250e-3
465	-6,2320920224565700e-3
466	-6,3323292492425800e-3
467	-6,4314484717364400e-3
468	-6,5293922762311450e-3
469	-6,6261035015351700e-3
470	-6,7215252466922100e-3
471	-6,8156013434509950e-3
472	-6,9082763630156450e-3
473	-6,9994952920827200e-3
474	-7,0892035490577050e-3
475	-7,1773476647806300e-3
476	-7,2638752839566550e-3
477	-7,3487345159041400e-3
478	-7,4318739341000500e-3
479	-7,5132432154309250e-3
480	-7,5927931627681400e-3
481	-7,6704755623721500e-3
482	-7,7462431738117650e-3
483	-7,8200496507836000e-3
484	-7,8918495587816850e-3
485	-7,9615988430090790e-3
486	-8,0292548409052500e-3
487	-8,0947760898363200e-3
488	-8,1581223027535600e-3
489	-8,2192540249893810e-3
490	-8,2781326752680240e-3
491	-8,3347217911436610e-3
492	-8,3889870206274740e-3
493	-8,4408946925453690e-3
494	-8,4904118087845760e-3
495	-8,5375073033623590e-3
496	-8,5821520672029750e-3
497	-8,6243182341614300e-3
498	-8,6639791627160300e-3
499	-8,7011097471794110e-3
500	-8,7356864659172800e-3
501	-8,7676881309490440e-3
502	-8,7970958152554810e-3
503	-8,8238905039462850e-3
504	-8,8480531477483150e-3
505	-8,8695681885783190e-3
506	-8,8884235927356400e-3
507	-8,9046080555685600e-3
508	-8,9181109318560050e-3
509	-8,9289234995380550e-3
510	-8,9370390010903460e-3
511	-8,9424522900292110e-3
512	-8,9451598223763610e-3
513	8,9451598223763610e-3
514	8,9424522900292110e-3
515	8,9370390010903460e-3
516	8,9289234995380550e-3
517	8,9181109318560050e-3
518	8,9046080555685600e-3
519	8,8884235927356400e-3
520	8,8695681885783190e-3
521	8,8480531477483150e-3
522	8,8238905039462850e-3
523	8,7970958152554810e-3
524	8,7676881309490440e-3
525	8,7356864659172800e-3
526	8,7011097471794110e-3
527	8,6639791627160300e-3
528	8,6243182341614300e-3
529	8,5821520672029750e-3
530	8,5375073033623590e-3
531	8,4904118087845760e-3
532	8,4408946925453690e-3
533	8,3889870206274740e-3
534	8,3347217911436610e-3
535	8,2781326752680240e-3
536	8,2192540249893810e-3
537	8,1581223027535600e-3
538	8,0947760898363200e-3
539	8,0292548409052500e-3
540	7,9615988430090790e-3
541	7,8918495587816850e-3
542	7,8200496507836000e-3
543	7,7462431738117650e-3
544	7,6704755623721500e-3
545	7,5927931627681400e-3
546	7,5132432154309250e-3
547	7,4318739341000500e-3
548	7,3487345159041400e-3
549	7,2638752839566550e-3
550	7,1773476647806300e-3
551	7,0892035490577050e-3
552	6,9994952920827200e-3
553	6,9082763630156450e-3
554	6,8156013434509950e-3
555	6,7215252466922100e-3
556	6,6261035015351700e-3
557	6,5293922762311450e-3
558	6,4314484717364400e-3
559	6,3323292492425800e-3
560	6,2320920224565700e-3
561	6,1307947602360250e-3
562	6,0284959882028950e-3
563	5,9252545216101900e-3
564	5,8211294443412250e-3
565	5,7161799140263450e-3
566	5,6104651646665850e-3
567	5,5040447592375600e-3
568	5,3969785783205650e-3
569	5,2893263173827950e-3
570	5,1811474713480250e-3
571	5,0725014978718950e-3
572	4,9634478250060780e-3
573	4,8540458565358850e-3
574	4,7443549566589130e-3
575	4,6344341075783170e-3

Coef #	Coefficient Value
576	4,5243418977567100e-3
577	4,4141366059086620e-3
578	4,3038762050876900e-3
579	4,1936183686073460e-3
580	4,0834204508378470e-3
581	3,9733390588103910e-3
582	3,8634300521859240e-3
583	3,7537489709635930e-3
584	3,6443510330122110e-3
585	3,5352906520539890e-3
586	3,4266214220855450e-3
587	3,3183962566563890e-3
588	3,2106673881690480e-3
589	3,1034862132032660e-3
590	2,9969032890820140e-3
591	2,8909684129992950e-3
592	2,7857306109716460e-3
593	2,6812378156433060e-3
594	2,5775368629845300e-3
595	2,4746737418493600e-3
596	2,3726935964105480e-3
597	2,2716405301701940e-3
598	2,1715575922482770e-3
599	2,0724866717198650e-3
600	1,9744684954557180e-3
601	1,8775426862784720e-3
602	1,7817477652804130e-3
603	1,6871211446584360e-3
604	1,5936991188210680e-3
605	1,5015166759080980e-3
606	1,4106074964587530e-3
607	1,3210041125919850e-3
608	1,2327379076720450e-3
609	1,1458389497708370e-3
610	1,0603359870796300e-3
611	9,7625651350435400e-4
612	8,9362676824381250e-4
613	8,1247166091487090e-4
614	7,3281477145293300e-4
615	6,5467842279709950e-4
616	5,7808367853976800e-4
617	5,0305021853266900e-4
618	4,2959633884346400e-4
619	3,5773907519488100e-4
620	2,8749420455512610e-4
621	2,1887615668780720e-4
622	1,5189801251884410e-4
623	8,6571556658577800e-5
624	2,2907279717153710e-5
625	-3,9085623279149610e-5
626	-9,9399252442146460e-5
627	-1,5802699163104190e-4
628	-2,1496350652513040e-4
629	-2,7020471361387040e-4
630	-3,2374777823389440e-4
631	-3,7559110239316400e-4
632	-4,2573432329476060e-4
633	-4,7417829303612970e-4
634	-5,2092507454409450e-4
635	-5,6597787244591100e-4
636	-6,0934103180698750e-4
637	-6,5102007948960100e-4
638	-6,9102172032639350e-4
639	-7,2935371157849700e-4

Coef #	Coefficient Value
640	-7,6602485902806300e-4
641	8,0104505655578600e-4
642	8,3442528375666700e-4
643	8,6617751390929650e-4
644	8,9631470926558100e-4
645	9,2485080945546600e-4
646	9,5180072722007060e-4
647	9,7718026613602800e-4
648	1,0010061160103870e-3
649	1,0232958338441730e-3
650	1,0440678410743190e-3
651	1,0633413819325920e-3
652	1,0811365181610850e-3
653	1,0974740543942100e-3
654	1,1123755320313420e-3
655	1,1258631690530700e-3
656	1,1379598570807350e-3
657	1,1486891568700730e-3
658	1,1580752925519680e-3
659	1,1661430294353880e-3
660	1,1729176690707690e-3
661	1,1784250628415020e-3
662	1,1826916065321440e-3
663	1,1857441072376680e-3
664	1,1876097809178440e-3
665	1,1883163316843900e-3
666	1,1878919433902330e-3
667	1,1863650153060240e-3
668	1,1837641581852870e-3
669	1,1801183720212010e-3
670	1,1754570428998680e-3
671	1,1698096960154200e-3
672	1,1632059885511800e-3
673	1,1556758000499310e-3
674	1,1472492294891640e-3
675	1,1379564405661050e-3
676	1,1278276550234160e-3
677	1,1168931605293980e-3
678	1,1051833075734240e-3
679	1,0927284332850520e-3
680	1,0795588568782010e-3
681	1,0657048556038330e-3
682	1,0511966606168980e-3
683	1,0360643900964910e-3
684	1,0203380430448350e-3
685	1,0040474297445790e-3
686	9,8722217067251400e-4
687	9,6989174214394400e-4
688	9,5208547141354700e-4
689	9,3383238324995840e-4
690	9,1516119514986000e-4
691	8,9610036547340950e-4
692	8,7667809201044900e-4
693	8,5692223230171550e-4
694	8,3686029869896850e-4
695	8,1651942936660300e-4
696	7,9592638676398950e-4
697	7,7510755325530950e-4
698	7,5408892753413600e-4
699	7,3289605035434600e-4
700	7,1155400216196750e-4
701	6,9008742526480550e-4
702	6,6852052145861900e-4
703	6,4687697763330650e-4

Coef #	Coefficient Value
704	6,2517996394886800e-4
705	6,0345216810040500e-4
706	5,8171579317559550e-4
707	5,5999247625413450e-4
708	5,3830328700499900e-4
709	5,1666877822680600e-4
710	4,9510898492628230e-4
711	4,7364335350426610e-4
712	4,5229073958127750e-4
713	4,3106943097409840e-4
714	4,0999714822829530e-4
715	3,8909103335715320e-4
716	3,6836764760397900e-4
717	3,4784293347385130e-4
718	3,2753221542219500e-4
719	3,0745025297166030e-4
720	2,8761124019708440e-4
721	2,6802874127778880e-4
722	2,4871569004884240e-4
723	2,2968444457655940e-4
724	2,1094678833176480e-4
725	1,9251390142613000e-4
726	1,7439635977956720e-4
727	1,5660414556657680e-4
728	1,3914664880465400e-4
729	1,2203269185287530e-4
730	1,0527052946041040e-4
731	8,8867825335203200e-5
732	7,2831653374563650e-5
733	5,7168548169515050e-5
734	4,1884506316022560e-5
735	2,6984964757674160e-5
736	1,2474801400564880e-5
737	-1,6416297019304470e-6
738	-1,5360496520469550e-5
739	-2,8678484003389880e-5
740	-4,1592792755171040e-5
741	-5,4101115008963200e-5
742	-6,6201632959158800e-5
743	-7,7893006105444910e-5
744	-8,9174369623037550e-5
745	-1,0004531116378850e-4
746	-1,1050586906665980e-4
747	-1,2055651618705740e-4
748	-1,3019815804235450e-4
749	-1,3943210818407590e-4
750	-1,4826008597798460e-4
751	-1,5668419239382670e-4
752	-1,6470690840351980e-4
753	-1,7233108395155490e-4
754	-1,7955993578371750e-4
755	-1,8639701070854740e-4
756	-1,9284618337957900e-4
757	-1,9891164424316180e-4
758	-2,0459789739171640e-4
759	-2,0990972538755550e-4
760	-2,1485218733324290e-4
761	-2,1943061159567880e-4
762	-2,2365059385031200e-4
763	-2,2751796132972940e-4
764	-2,3103877007019660e-4
765	-2,3421928008696620e-4
766	-2,3706595356421380e-4
767	-2,3958543985834340e-4

Coef #	Coefficient Value
768	-2,4178457457919800e-4
769	2,4367037437341020e-4
770	2,4525003217524440e-4
771	2,4653081794176570e-4
772	2,4752008219951220e-4
773	2,4822543333608940e-4
774	2,4865471770248810e-4
775	2,4881540457247040e-4
776	2,4871461830381660e-4
777	2,4836046092637280e-4
778	2,4776199025177680e-4
779	2,4692741569938220e-4
780	2,4586407616017000e-4
781	2,4457975717687980e-4
782	2,4308272452941820e-4
783	2,4138114593372380e-4
784	2,3948307023621970e-4
785	2,3739654799291240e-4
786	2,3512963563507800e-4
787	2,3269036653525150e-4
788	2,3008674810081220e-4
789	2,2732672676686290e-4
790	2,2441818635854840e-4
791	2,2136894706917190e-4
792	2,1818676491655660e-4
793	2,1487932080488700e-4
794	2,1145421910491280e-4
795	2,0791896735659270e-4
796	2,0428097498957360e-4
797	2,0054754547108610e-4
798	1,9672587556804270e-4
799	1,9282304696531290e-4
800	1,8884602526638050e-4
801	1,8480164638953740e-4
802	1,8069661584612290e-4
803	1,7653750646560020e-4
804	1,7233075773584680e-4
805	1,6808266356446820e-4
806	1,6379937154642740e-4
807	1,5948687909058970e-4
808	1,5515103314094370e-4
809	1,5079752791692290e-4
810	1,4643190426323540e-4
811	1,4205953760578120e-4
812	1,3768563787408340e-4
813	1,3331525983885840e-4
814	1,2895330270761850e-4
815	1,2460449089034750e-4
816	1,2027337376184880e-4
817	1,1596433966772030e-4
818	1,1168161614264270e-4
819	1,0742926070674010e-4
820	1,0321116057304920e-4
821	9,9031035415475500e-5
822	9,4892437636254000e-5
823	9,0798755482139240e-5
824	8,6753213062403400e-5
825	8,2758867628623800e-5
826	7,8818609806320000e-5
827	7,4935171411178040e-5
828	7,1111125753722860e-5
829	6,7348886525882700e-5
830	6,3650708053788550e-5
831	6,0018691972791050e-5
832	5,6454787620185350e-5
833	5,2960794000140600e-5
834	4,9538360176954340e-5
835	4,6188991961724560e-5
836	4,2914052387856640e-5
837	3,9714765485539260e-5
838	3,6592216716540970e-5
839	3,3547358765259040e-5
840	3,0581012081916960e-5
841	2,7693871041862060e-5
842	2,4886504491416160e-5
843	2,2159361597292710e-5
844	1,9512772389215240e-5
845	1,6946953848107710e-5
846	1,4462010500550970e-5
847	1,2057941408635810e-5
848	9,7346407524871100e-6
849	7,4919036559427900e-6
850	5,3294268968052800e-6
851	3,2468187066945930e-6
852	1,2435990827839660e-6
853	-6,8080315386959900e-7
854	-2,5270494199802070e-6
855	-4,2958677847708840e-6
856	-5,9880523825897000e-6
857	-7,6044740757265500e-6
858	-9,1460801506922300e-6
859	-1,0613876973541650e-5
860	-1,2008929083235120e-5
861	-1,3332356590263080e-5
862	-1,4585334740507920e-5
863	-1,5769086921577900e-5
864	-1,6884884022311310e-5
865	-1,7934037335534700e-5
866	-1,8917898053528080e-5
867	-1,9837853265502040e-5
868	-2,0695325418576370e-5
869	-2,1491764461832140e-5
870	-2,2228647280994560e-5
871	-2,2907473117403740e-5
872	-2,3529763128539150e-5
873	-2,4097055300584120e-5
874	-2,4610903957661350e-5
875	-2,5072874052447260e-5
876	-2,5484540698002960e-5
877	-2,5847484577439320e-5
878	-2,6163291550639320e-5
879	-2,6433548592482960e-5
880	-2,6659843389908530e-5
881	-2,6843759539035120e-5
882	-2,6986876191054700e-5
883	-2,7090765064735940e-5
884	-2,7156990105794550e-5
885	-2,7187102980536760e-5
886	-2,7182642746157720e-5
887	-2,7145133103663890e-5
888	-2,7076082153012070e-5
889	-2,6976979753245170e-5
890	-2,6849297239997120e-5
891	-2,6694483850464800e-5
892	-2,6513966535124460e-5
893	-2,6309149390520640e-5
894	-2,6081413415833200e-5
895	-2,5832111724562040e-5
896	-2,5562569399199020e-5
897	2,5274085082324960e-5
898	2,4967930863542620e-5
899	2,4645348200903720e-5
900	2,4307547722489910e-5
901	2,3955708939763280e-5
902	2,3590980216497750e-5
903	2,3214478371918170e-5
904	2,2827288605503470e-5
905	2,2430463239517170e-5
906	2,2025021652894630e-5
907	2,1611950084231910e-5
908	2,1192201630574280e-5
909	2,0766696417773940e-5
910	2,0336321557464510e-5
911	1,9901930030141080e-5
912	1,9464340744880170e-5
913	1,9024340970099240e-5
914	1,8582686308483550e-5
915	1,8140097714618200e-5
916	1,7697261563607500e-5
917	1,7254834143068570e-5
918	1,6813441716539560e-5
919	1,6373677426466900e-5
920	1,5936101341640500e-5
921	1,5501244597782600e-5
922	1,5069609527679710e-5
923	1,4641668383496450e-5
924	1,4217863365843780e-5
925	1,3798608540965960e-5
926	1,3384289978242060e-5
927	1,2975266858030310e-5
928	1,2571871574469220e-5
929	1,2174410889256530e-5
930	1,1783166036693560e-5
931	1,1398393880881060e-5
932	1,1020327046334650e-5
933	1,0649175634328590e-5
934	1,0285127334608450e-5
935	9,9283481667264540e-6
936	9,5789825988253100e-6
937	9,2371554195468400e-6
938	8,9029718900122000e-6
939	8,5765192154768100e-6
940	8,2578666429227760e-6
941	7,9470661364365490e-6
942	7,6441525098592000e-6
943	7,3491456697144900e-6
944	7,0620507545154100e-6
945	6,7828589565804950e-6
946	6,5115476278604000e-6
947	6,2480817863072800e-6
948	5,9924142424804800e-6
949	5,7444868785028750e-6
950	5,5042307535458100e-6
951	5,2715671455405850e-6
952	5,0464076705003400e-6
953	4,8286558658894680e-6
954	4,6182073040543900e-6
955	4,4149505043312320e-6
956	4,2187670209852250e-6
957	4,0295324657699800e-6
958	3,8471166218015830e-6
959	3,6713849262294680e-6

Coef #	Coefficient Value
960	3,5021985520037540e-6
961	3,3394147240855820e-6
962	3,1828867865974640e-6
963	3,0324653639203880e-6
964	2,8879984579501380e-6
965	2,7493324266391140e-6
966	2,6163120571644820e-6
967	2,4887811970856540e-6
968	2,3665828120950560e-6
969	2,2495596253776460e-6
970	2,1375541686662320e-6
971	2,0304092660990760e-6
972	1,9279680902322050e-6
973	1,8300749104526120e-6
974	1,7365751553971340e-6
975	1,6473160377545360e-6
976	1,5621465913980060e-6
977	1,4809178634979940e-6
978	1,4034829433780260e-6
979	1,3296974052300520e-6
980	1,2594193438633260e-6
981	1,1925097184646100e-6
982	1,1288323637155430e-6
983	1,0682538906146330e-6
984	1,0106437128213760e-6
985	9,5587472540498280e-7
986	9,0382333224895500e-7
987	8,5436937020293650e-7
988	8,0739610368280600e-7
989	7,6279018172077200e-7
990	7,2044165259013450e-7
991	6,8024432849991400e-7
992	6,4209580087853050e-7
993	6,0589741192019800e-7
994	5,7155423388793000e-7
995	5,3897464223335550e-7
996	5,0807031294631050e-7
997	4,7875659112606760e-7
998	4,5095249563698380e-7
999	4,2458045296159330e-7
1 000	3,9956628121669190e-7
1 001	3,7583910465412180e-7
1 002	3,5333134588843600e-7
1 003	3,3197864040473180e-7
1 004	3,1171981968268900e-7
1 005	2,9249662541757600e-7
1 006	2,7425370308549580e-7
1 007	2,5693874615777660e-7
1 008	2,4050245877923400e-7
1 009	2,2489759047553620e-7
1 010	2,1007899706006800e-7
1 011	1,9600594587757850e-7
1 012	1,8264184311018150e-7
1 013	1,6994592926571220e-7
1 014	1,5787424998123710e-7
1 015	1,4640931821191640e-7
1 016	1,3555636137971380e-7
1 017	1,2523109645615350e-7
1 018	1,1527363411865760e-7
1 019	1,0586659510721950e-7
1 020	9,7483190494874640e-8
1 021	8,9326871281374790e-8
1 022	7,9525034321375090e-8
1 023	7,0950903150874990e-8
1 024	7,1279389866041690e-8

D.10 VQ Tables

D.10.1 ADPCM Coefficients

Each vector consists of four elements and the Codebook has 2^{12} = 4 096 vectors. Each entry represents an element multiplied by 2^{13}. Therefore, the actual value of each element is calculated as follows:

$$ActualElementValue = \frac{Entry}{2^{13}}$$

For example, the first entry in the table gives:

$$\frac{9928}{2^{13}} = 1.2119140625$$

Due to its extensive size, this table is not included here.

D.10.2 High Frequency Subbands

Each vector consists of 32 elements and the Codebook has 2^{10} = 1 024 vectors. Each entry is 16 bits, representing two vector elements, so it takes 16 entries in the table to represent one vector of 32 elements. Each entry is first be split into two 8-bit integers and then each divided by 24 to give two vector elements.

- Due to its extensive size, this table is not included here.

D.11 Look-up Table for Downmix Scale Factors

DmixTblIndex	LogAbsValues (dB)	AbsValues	DmixTable	InvDmixTblIndex	InvDmixTbl
0	-60,0000	0,001000	33	N/A	N/A
1	-59,5000	0,001059	35	N/A	N/A
2	-59,0000	0,001122	37	N/A	N/A
3	-58,5000	0,001189	39	N/A	N/A
4	-58,0000	0,001259	41	N/A	N/A
5	-57,5000	0,001334	44	N/A	N/A
6	-57,0000	0,001413	46	N/A	N/A
7	-56,5000	0,001496	49	N/A	N/A
8	-56,0000	0,001585	52	N/A	N/A
9	-55,5000	0,001679	55	N/A	N/A
10	-55,0000	0,001778	58	N/A	N/A
11	-54,5000	0,001884	62	N/A	N/A
12	-54,0000	0,001995	65	N/A	N/A
13	-53,5000	0,002113	69	N/A	N/A
14	-53,0000	0,002239	73	N/A	N/A
15	-52,5000	0,002371	78	N/A	N/A
16	-52,0000	0,002512	82	N/A	N/A
17	-51,5000	0,002661	87	N/A	N/A
18	-51,0000	0,002818	92	N/A	N/A
19	-50,5000	0,002985	98	N/A	N/A
20	-50,0000	0,003162	104	N/A	N/A
21	-49,5000	0,003350	110	N/A	N/A
22	-49,0000	0,003548	116	N/A	N/A
23	-48,5000	0,003758	123	N/A	N/A
24	-48,0000	0,003981	130	N/A	N/A
25	-47,5000	0,004217	138	N/A	N/A
26	-47,0000	0,004467	146	N/A	N/A
27	-46,5000	0,004732	155	N/A	N/A
28	-46,0000	0,005012	164	N/A	N/A
29	-45,5000	0,005309	174	N/A	N/A

DmixTblIndex	LogAbsValues (dB)	AbsValues	DmixTable	InvDmixTblIndex	InvDmixTbl
30	-45,0000	0,005623	184	N/A	N/A
31	-44,5000	0,005957	195	N/A	N/A
32	-44,0000	0,006310	207	N/A	N/A
33	-43,5000	0,006683	219	N/A	N/A
34	-43,0000	0,007079	232	N/A	N/A
35	-42,5000	0,007499	246	N/A	N/A
36	-42,0000	0,007943	260	N/A	N/A
37	-41,5000	0,008414	276	N/A	N/A
38	-41,0000	0,008913	292	N/A	N/A
39	-40,5000	0,009441	309	N/A	N/A
40	-40,0000	0,010000	328	0	6553600
41	-39,5000	0,010593	347	1	6186997
42	-39,0000	0,011220	368	2	5840902
43	-38,5000	0,011885	389	3	5514167
44	-38,0000	0,012589	413	4	5205710
45	-37,5000	0,013335	437	5	4914507
46	-37,0000	0,014125	463	6	4639593
47	-36,5000	0,014962	490	7	4380059
48	-36,0000	0,015849	519	8	4135042
49	-35,5000	0,016788	550	9	3903731
50	-35,0000	0,017783	583	10	3685360
51	-34,5000	0,018836	617	11	3479204
52	-34,0000	0,019953	654	12	3284581
53	-33,5000	0,021135	693	13	3100844
54	-33,0000	0,022387	734	14	2927386
55	-32,5000	0,023714	777	15	2763630
56	-32,0000	0,025119	823	16	2609035
57	-31,5000	0,026607	872	17	2463088
58	-31,0000	0,028184	924	18	2325305
59	-30,5000	0,029854	978	19	2195230
60	-30,0000	0,031623	1036	20	2072430
61	-29,7500	0,032546	1066	21	2013631
62	-29,5000	0,033497	1098	22	1956500
63	-29,2500	0,034475	1130	23	1900990
64	-29,0000	0,035481	1163	24	1847055
65	-28,7500	0,036517	1197	25	1794651
66	-28,5000	0,037584	1232	26	1743733
67	-28,2500	0,038681	1268	27	1694260
68	-28,0000	0,039811	1305	28	1646190
69	-27,7500	0,040973	1343	29	1599484
70	-27,5000	0,042170	1382	30	1554103
71	-27,2500	0,043401	1422	31	1510010
72	-27,0000	0,044668	1464	32	1467168
73	-26,7500	0,045973	1506	33	1425542
74	-26,5000	0,047315	1550	34	1385096
75	-26,2500	0,048697	1596	35	1345798
76	-26,0000	0,050119	1642	36	1307615
77	-25,7500	0,051582	1690	37	1270515
78	-25,5000	0,053088	1740	38	1234468
79	-25,2500	0,054639	1790	39	1199444
80	-25,0000	0,056234	1843	40	1165413
81	-24,7500	0,057876	1896	41	1132348
82	-24,5000	0,059566	1952	42	1100221
83	-24,2500	0,061306	2009	43	1069005
84	-24,0000	0,063096	2068	44	1038676
85	-23,7500	0,064938	2128	45	1009206
86	-23,5000	0,066834	2190	46	980573
87	-23,2500	0,068786	2254	47	952752
88	-23,0000	0,070795	2320	48	925721
89	-22,7500	0,072862	2388	49	899456
90	-22,5000	0,074989	2457	50	873937
91	-22,2500	0,077179	2529	51	849141
92	-22,0000	0,079433	2603	52	825049
93	-21,7500	0,081752	2679	53	801641

DmixTblIndex	LogAbsValues (dB)	AbsValues	DmixTable	InvDmixTblIndex	InvDmixTbl
94	-21,5000	0,084140	2757	54	778897
95	-21,2500	0,086596	2838	55	756798
96	-21,0000	0,089125	2920	56	735326
97	-20,7500	0,091728	3006	57	714463
98	-20,5000	0,094406	3093	58	694193
99	-20,2500	0,097163	3184	59	674497
100	-20,0000	0,100000	3277	60	655360
101	-19,7500	0,102920	3372	61	636766
102	-19,5000	0,105925	3471	62	618700
103	-19,2500	0,109018	3572	63	601146
104	-19,0000	0,112202	3677	64	584090
105	-18,7500	0,115478	3784	65	567518
106	-18,5000	0,118850	3894	66	551417
107	-18,2500	0,122321	4008	67	535772
108	-18,0000	0,125893	4125	68	520571
109	-17,7500	0,129569	4246	69	505801
110	-17,5000	0,133352	4370	70	491451
111	-17,2500	0,137246	4497	71	477507
112	-17,0000	0,141254	4629	72	463959
113	-16,7500	0,145378	4764	73	450796
114	-16,5000	0,149624	4903	74	438006
115	-16,2500	0,153993	5046	75	425579
116	-16,0000	0,158489	5193	76	413504
117	-15,7500	0,163117	5345	77	401772
118	-15,5000	0,167880	5501	78	390373
119	-15,2500	0,172783	5662	79	379297
120	-15,0000	0,177828	5827	80	368536
121	-14,8750	0,180406	5912	81	363270
122	-14,7500	0,183021	5997	82	358080
123	-14,6250	0,185674	6084	83	352964
124	-14,5000	0,188365	6172	84	347920
125	-14,3750	0,191095	6262	85	342949
126	-14,2500	0,193865	6353	86	338049
127	-14,1250	0,196675	6445	87	333219
128	-14,0000	0,199526	6538	88	328458
129	-13,8750	0,202418	6633	89	323765
130	-13,7500	0,205353	6729	90	319139
131	-13,6250	0,208329	6827	91	314579
132	-13,5000	0,211349	6925	92	310084
133	-13,3750	0,214412	7026	93	305654
134	-13,2500	0,217520	7128	94	301287
135	-13,1250	0,220673	7231	95	296982
136	-13,0000	0,223872	7336	96	292739
137	-12,8750	0,227117	7442	97	288556
138	-12,7500	0,230409	7550	98	284433
139	-12,6250	0,233749	7659	99	280369
140	-12,5000	0,237137	7771	100	276363
141	-12,3750	0,240575	7883	101	272414
142	-12,2500	0,244062	7997	102	268522
143	-12,1250	0,247600	8113	103	264685
144	-12,0000	0,251189	8231	104	260904
145	-11,8750	0,254830	8350	105	257176
146	-11,7500	0,258523	8471	106	253501
147	-11,6250	0,262271	8594	107	249879
148	-11,5000	0,266073	8719	108	246309
149	-11,3750	0,269929	8845	109	242790
150	-11,2500	0,273842	8973	110	239321
151	-11,1250	0,277811	9103	111	235901
152	-11,0000	0,281838	9235	112	232531
153	-10,8750	0,285924	9369	113	229208
154	-10,7500	0,290068	9505	114	225933
155	-10,6250	0,294273	9643	115	222705
156	-10,5000	0,298538	9783	116	219523
157	-10,3750	0,302866	9924	117	216386

DmixTblIndex	LogAbsValues (dB)	AbsValues	DmixTable	InvDmixTblIndex	InvDmixTbl
158	-10,2500	0,307256	10068	118	213295
159	-10,1250	0,311709	10214	119	210247
160	-10,0000	0,316228	10362	120	207243
161	-9,8750	0,320812	10512	121	204282
162	-9,7500	0,325462	10665	122	201363
163	-9,6250	0,330179	10819	123	198486
164	-9,5000	0,334965	10976	124	195650
165	-9,3750	0,339821	11135	125	192855
166	-9,2500	0,344747	11297	126	190099
167	-9,1250	0,349744	11460	127	187383
168	-9,0000	0,354813	11627	128	184706
169	-8,8750	0,359956	11795	129	182066
170	-8,7500	0,365174	11966	130	179465
171	-8,6250	0,370467	12139	131	176901
172	-8,5000	0,375837	12315	132	174373
173	-8,3750	0,381285	12494	133	171882
174	-8,2500	0,386812	12675	134	169426
175	-8,1250	0,392419	12859	135	167005
176	-8,0000	0,398107	13045	136	164619
177	-7,8750	0,403878	13234	137	162267
178	-7,7500	0,409732	13426	138	159948
179	-7,6250	0,415671	13621	139	157663
180	-7,5000	0,421697	13818	140	155410
181	-7,3750	0,427809	14018	141	153190
182	-7,2500	0,434010	14222	142	151001
183	-7,1250	0,440301	14428	143	148844
184	-7,0000	0,446684	14637	144	146717
185	-6,8750	0,453158	14849	145	144621
186	-6,7500	0,459727	15064	146	142554
187	-6,6250	0,466391	15283	147	140517
188	-6,5000	0,473151	15504	148	138510
189	-6,3750	0,480010	15729	149	136531
190	-6,2500	0,486968	15957	150	134580
191	-6,1250	0,494026	16188	151	132657
192	-6,0000	0,501187	16423	152	130762
193	-5,8750	0,508452	16661	153	128893
194	-5,7500	0,515822	16902	154	127052
195	-5,6250	0,523299	17147	155	125236
196	-5,5000	0,530884	17396	156	123447
197	-5,3750	0,538580	17648	157	121683
198	-5,2500	0,546387	17904	158	119944
199	-5,1250	0,554307	18164	159	118231
200	-5,0000	0,562341	18427	160	116541
201	-4,8750	0,570493	18694	161	114876
202	-4,7500	0,578762	18965	162	113235
203	-4,6250	0,587151	19240	163	111617
204	-4,5000	0,595662	19519	164	110022
205	-4,3750	0,604296	19802	165	108450
206	-4,2500	0,613056	20089	166	106901
207	-4,1250	0,621942	20380	167	105373
208	-4,0000	0,630957	20675	168	103868
209	-3,8750	0,640103	20975	169	102383
210	-3,7500	0,649382	21279	170	100921
211	-3,6250	0,658795	21587	171	99479
212	-3,5000	0,668344	21900	172	98057
213	-3,3750	0,678032	22218	173	96656
214	-3,2500	0,687860	22540	174	95275
215	-3,1250	0,697831	22867	175	93914
216	-3,0000	0,707107	23170	176	92682
217	-2,8750	0,718208	23534	177	91249
218	-2,7500	0,728618	23875	178	89946
219	-2,6250	0,739180	24221	179	88660
220	-2,5000	0,749894	24573	180	87394
221	-2,3750	0,760764	24929	181	86145

DmixTblIndex	LogAbsValues (dB)	AbsValues	DmixTable	InvDmixTblIndex	InvDmixTbl
222	-2,2500	0,771792	25290	182	84914
223	-2,1250	0,782979	25657	183	83701
224	-2,0000	0,794328	26029	184	82505
225	-1,8750	0,805842	26406	185	81326
226	-1,7500	0,817523	26789	186	80164
227	-1,6250	0,829373	27177	187	79019
228	-1,5000	0,841395	27571	188	77890
229	-1,3750	0,853591	27970	189	76777
230	-1,2500	0,865964	28376	190	75680
231	-1,1250	0,878517	28787	191	74598
232	-1,0000	0,891251	29205	192	73533
233	-0,8750	0,904170	29628	193	72482
234	-0,7500	0,917276	30057	194	71446
235	-0,6250	0,930572	30493	195	70425
236	-0,5000	0,944061	30935	196	69419
237	-0,3750	0,957745	31383	197	68427
238	-0,2500	0,971628	31838	198	67450
239	-0,1250	0,985712	32300	199	66486
240	0	1,000000	32768	200	65536

Annex E (normative): DTS and DTS-HD formats in ISO Media Files

E.1 Overview

The audio format defined in the present specification will be broadly referred to as DTS-HD audio for the purposes of this Annex. DTS-HD audio can be stored in a format compatible with the ISO Media File Format defined in ISO/IEC 14496-12 [5]. This annex defines the signalling and encapsulation of DTS-HD audio in ISO Media Files.

E.2 Signalling

E.2.1 Track Header

DTS-HD tracks shall be compliant with audio tracks as defined in ISO/IEC 14496-12 [5]. The following rules shall be applied to the boxes within the Media Box in a DTS-HD track:

- The `handler_type` field in the Handler Reference Box shall be set to `'soun'`
- The Media Information Header Box shall be of type `'smhd'`
- The Sample Description Box shall be derived from AudioSampleEntry as described in clause E.3.
- The `timescale` parameter in the Media Header Box shall be set to `samplerate` or an integer multiple thereof. Configuration of `samplerate` is described in clause E.2.2.2.
- All DTS-HD samples are random access points, so the Sync Sample Box shall not be present in a DTS-HD track.

E.2.2 SampleDescription Box

E.2.2.1 Overview of SampleDescription Box

The DTS SampleEntry box is derived from the AudioSampleEntry box defined in ISO/IEC 14496-12 [5]. The dts-specific SampleEntry box shall be identified by a unique codingname value (see Table E-1). The codingname corresponds to the DTS-HD stream composition as shown in Table E-2.

Table E-1: Defined Audio Formats

codingname	Description
dtsc	DTS formats prior to DTS-HD
dtsh	DTS-HD audio formats
dtsl	DTS-HD Lossless formats
dtse	DTS Low Bit Rate (LBR) formats

E.2.2.2 DTS_SampleEntry

DTS_SampleEntry extends the AudioSampleEntry box defined in ISO/IEC 14496-12 [5]:

```
Class DTS_SampleEntry() extends AudioSampleEntry (codingname) {
    DTSSpecificBox()           // 'ddts' box
}
```

For DTS_SampleEntry(), the following values inherited from AudioSampleEntry are set as follows:

codingname is according to table E-1.

channelcount is set to the number of decodable output channels in basic playback, as described in the 'ddts' configuration box. Additional channel count as a result of future feature enhancements are defined in a box following the 'ddts' box, where ReservedBox() is the placeholder.

samplesize is always set to 16.

samplerate is set according to DTSSamplingFrequency of either:

- 48 000 for original sampling frequencies of 24 000 Hz, 48 000 Hz, 96 000 Hz or 192 000 Hz;
- 44 100 for original sampling frequencies of 22 050 Hz, 44 100 Hz, 88 200 Hz or 176 400 Hz;
- 32 000 for original sampling frequencies of 16 000 Hz, 32 000 Hz, 64 000 Hz or 128 000 Hz.

E.2.2.3 DTSSpecificBox

E.2.2.3.1 Syntax of DTSSpecificBox

The syntax and semantics of the DTSSpecificBox ('ddts') are shown below:

```
class DTSSpecificBox extends Box ('ddts'){
    unsigned int(32)    size;
    unsigned char       type[4]  = 'ddts';
    unsigned int(32)    DTSSamplingFrequency;
    unsigned int(32)    maxBitrate;
    unsigned int(32)    avgBitrate;
    unsigned char       pcmSampleDepth;         // value is 16 or 24 bits
    bit(2)              FrameDuration;          // 0 = 512, 1 = 1024, 2 = 2048, 3 = 4096
    bit(5)              StreamConstruction;     // Table E-2
    bit(1)              CoreLFEPresent;         // 0 = none; 1 = LFE exists
    bit(6)              CoreLayout;             // Table E-3
    bit(14)             CoreSize;
    bit(1)              StereoDownmix           // 0 = none; 1 = embedded downmix present
    bit(3)              RepresentationType;     // Table E-4
    bit(16)             ChannelLayout;          // Table E-5
    bit(1)              MultiAssetFlag          // 0 = single asset, 1 = multiple asset
    bit(1)              LBRDurationMod          // 0 = ignore, 1 = Special LBR duration modifier
    bit(1)              ReservedBoxPresent      // 0 = NoReservedBox, 1 = NoReservedBox present
    bit(5)              Reserved                // Reserved bits are set to 0
    ReservedBox()                               // optional, for future expansion
};
```

E.2.2.3.2 Semantics

DTSSamplingFrequency: The maximum sampling frequency stored in the compressed audio stream.

pcmSampleDepth: The bit depth of the rendered audio. For DTS formats this is usually 24-bits.

maxBitrate: The peak bit rate, in bits per second, of the audio elementary stream for the duration of the track, including the core substream (if present) and all extension substreams. If the stream is a constant bit rate, this parameter has the same value as avgBitrate. If the maximum bit rate is unknown, this parameter is set to 0.

avgBitrate: The average bit rate, in bits per second, of the audio elementary stream for the duration of the track, including the core substream and any extension substream that may be present.

FrameDuration: This code represents the number of audio samples decoded in a complete audio access unit at **DTSSampling Frequency**.

StreamConstructon: Provides complete information on the existence and of location of extensions in any synchronized frame. See Table E-2. For any stream type not listed in Table E-2, this parameter is set to 0 and the coding name defaults to dtsh.

Table E-2: StreamConstruction

StreamConstruction	Core substream				Extension substream						codingname
	Core	XCH	X96	XXCH	Core	XXCH	X96	XBR	XLL	LBR	
1	✓										dtsc
2	✓	✓									dtsc
3	✓			✓							dtsh
4	✓		✓								dtsc
5	✓					✓					dtsh
6	✓							✓			dtsh
7	✓	✓						✓			dtsh
8	✓			✓				✓			dtsh
9	✓					✓		✓			dtsh
10	✓						✓				dtsh
11	✓	✓					✓				dtsh
12	✓			✓			✓				dtsh
13	✓					✓	✓				dtsh
14	✓								✓		dtsh
15	✓	✓							✓		dtsh
16	✓		✓						✓		dtsh
17									✓		dtsl
18										✓	dtse
19					✓						dtsh
20					✓	✓					dtsh
21					✓				✓		dtsh

CoreLFEPresent: Indicates the presence of an LFE channel in the core. If no core substream exists, this value is ignored.

CoreLayout: This parameter represents the channel layout of the core within the core substream and is set according to Table E-3. If no core substream exists, this parameter is ignored and **ChannelLayout** or RepresentationType is used to determine channel configuration.

Table E-3: CoreLayout

Core Layout	Description
0	Mono (1/0)
2	Stereo (2/0)
4	LT,RT (2/0)
5	L, C, R (3/0)
6	L, R, S (2/1)
7	L, C, R, S (3/1)
8	L, R, LS, RS (2/2)
9	L, C, R, LS, RS (3/2)
31	use ChannelLayout

For streams where StreamConstruction is undefined (i.e. StreamConstruction = 0), or a DTS core component only exists in the extension substream (e.g. StreamConstruction = 19, 20 or 21), CoreLayout is set to 31.

All undefined values for CoreLayout are reserved for future use.

CoreSize: The size of a core substream AU in bytes. If no core substream exists **CoreSize**=0 and parameters **CoreLayout** and **CoreLFEPresent** are ignored.

StereoDownmix: Indicates the presence of an embedded stereo downmix in the stream This parameter is not valid for stereo or mono streams.

RepresentationType: This indicates special properties of the audio presentation, as indicated in Table E-4. This parameter is only valid when all flags in **ChannelLayout** are set to 0. If **ChannelLayout** $\neq 0$, this value is ignored.

Table E-4: RepresentationType

RepresentationType	Description
0	Audio asset designated for mixing with another audio asset
2	Lt/Rt Encoded for matrix surround decoding; it implies that total number of encoded channels is 2
3	Audio processed for headphone playback; it implies that total number of encoded channels is 2
1 and 4 through 7	Reserved

ChannelLayout: Provides complete information on channels coded in the audio stream including core and extensions. The binary masks of the channels present, as shown in Table E-5, are added together to create **ChannelLayout**.

Table E-5: ChannelLayout

Bit Masks	Loudspeaker Location Description	Number of Channels
0001h	Centre in front of listener	1
0002h	Left/Right in front	2
0004h	Left/Right surround on side in rear	2
0008h	Low frequency effects subwoofer	1
0010h	Centre surround in rear	1
0020h	Left/Right height in front	2
0040h	Left/Right surround in rear	2
0080h	Centre Height in front	1
0100h	Over the listener's head	1
0200h	Between left/right and centre in front	2
0400h	Left/Right on side in front	2
0800h	Left/Right surround on side	2
1000h	Second low frequency effects subwoofer	1
2000h	Left/Right height on side	2
4000h	Centre height in rear	1
8000h	Left/Right height in rear	2

MultiAssetFlag: This flag is set if the stream contains more than one asset. This also implies that a DTS extension substream is present. Multiple asset streams use the 'dtsh' coding type. When multiple assets exist, the remaining parameters in the DTSSpecificBox only reflect the coding parameters of the first asset.

LBRDurationMod: This flag indicates a special case of the LBR coding bandwidth, resulting in 1/3 or 2/3 band limiting. The result of this is the LBR frame duration is 50 % larger than indicated in FrameDuration. For example, when this flag is set to 1, the FrameDuration is 6 144 samples instead of 4 096 samples.

Reserved: These bits are reserved for future definition. ISO media files created according to this version of specification will have these bits set to 0.

E.2.2.3.3 ReservedBox

The reserved box is optional and serves as a placeholder for future expansion. Additional private boxes may follow the 'ddts' box in the DTS_SampleEntry(). Playback devices not equipped to support these additional extensions depend on the 'ddts' box for basic playback capability.

E.3 Storage of DTS-HD Elementary Streams

One DTS-HD audio frame constitutes one sample. Samples shall be stored in the 'mdat' box in the same order in which they are intended to be played back.

E.4 Restrictions on DTS Formats

The following conditions shall remain constant in a core substream for seamless playback:

- Duration of Synchronized Frame
- Sampling Frequency
- Audio Channel Arrangement
- Low Frequency Effects flag
- Extension assignment as indicated in StreamConstruction.

The following conditions shall remain constant in an Extension substream for seamless playback:

- Duration of Synchronized Frame
- Sampling Frequency
- Audio Channel Arrangement including LFE
- Embedded stereo flag
- Extensions assignment indicated in **StreamConstruction**

E.5 Implementation of DTS Sample Entry

The information needed to derive the elements of the DTS Sample Entry box and boxes contained within it, may be extracted from the respective elementary stream. DTS has tools available to implementers that will analyse DTS elementary streams and extract the information necessary to populate these parameters. DTS document #9302J81100, describes the function calls and return structures. To obtain this tool and additional documentation, please direct all document requests to DTS Licensing at LicensingAdministration@dts.com.

Annex F (normative): Application of DTS formats to MPEG-2 Streams

F.1 Overview of Annex F

This annex specifies how DTS and DTS-HD audio is applied in MPEG-2 systems and provides additional information and references to the usage of DTS and DTS-HD elementary streams in DVB broadcast applications. While the use of DTS formats in DVB broadcast is optional, if they are used, the present document should be followed.

This Annex is informative in that it is not required to use DTS or DTS-HD audio in an MPEG-2 system. However, if the audio formats specified in the present document are implemented in MPEG-2 systems, this annex is to be followed.

Additional information pertaining to DTS formats in other MPEG-2 TS environments may be available at www.dts.com.

F.2 Buffering Model

The DTS buffering model is designed in accordance with ISO/IEC 13818-1 [6]. Refer to the derivation of BS_n for audio elementary streams.

- For DTS core streams, the main audio buffer size (BS_n) has a fixed value of 9 088 bytes, with a drain rate (Rx_n) of 2 Mbps. The fixed value above (9 088 bytes) was calculated from a double buffer (2 × 4 096 bytes) plus jitter (384 bytes) + packet bursts (512 bytes).
- For DTS-HD Lossless formats, the value of BS_n has a fixed value of 66 432 bytes, with an Rx_n value of 32 Mbps.
- For all other DTS-HD formats, the value of BS_n has a fixed value of 17 814 bytes, with an Rx_n value of 8 Mbps.

F.3 Signalling

F.3.1 PSI Signalling in the PMT

F.3.1.1 Overview of PSI Signalling for DTS and DTS-HD

Two related generations of DTS formats exist, the original DTS core format and the expanded DTS-HD format. As a result of this second generation of DTS formats, a new DTS-HD audio descriptor was created to accommodate the expanded feature set. This new structures can accommodate core only formats as well as extension only and core + extension combinations. If an MPEG-2 system supports DTS-HD, all DTS formats broadcast in that system may use the DTS-HD signalling as described in clause G.3 in ETSI EN 300 468 [1].

F.3.1.2 Stream Type

In DVB systems, and systems that follow DVB convention, DTS and DTS-HD elementary streams are signalled as private_stream_1 and therefore use a stream_type = 0x06, consistent with ETSI TS 101 154 [2], clause 4.1.6.1 and in accordance with Recommendation ITU-T H.222.0/ISO/IEC 13818-1 [6].

In systems that follow ATSC convention, such as SCTE, DTS and DTS-HD have been assigned a value in the ATSC registry, therefore stream_type is to be set to 0x88.

F.4 Elementary Stream Encapsulation

F.4.1 Stream ID

All DTS and DTS-HD elementary streams use a stream_id = 0xBD, indicating private stream 1, in accordance with Recommendation ITU-T H.222.0/ISO/IEC 13818-1 [6]. Multiple DTS/DTS-HD streams may share the same value of stream_id since each stream is carried with a unique PID value. The mapping of values of PID to stream_type is indicated in the transport stream PMT.

F.4.2 Calculation of PTS from the elementary stream

F.4.2.1 Calculating Time Duration

The time duration of one audio access unit can be calculated by dividing the audio frame duration in samples by the audio sampling frequency.

F.4.2.2 Frame Duration from Core Substream Metadata

In the case of a core substream, the audio frame duration for a normal (non-termination) frame is NBLKS, (described in clause 5.4.2) and the audio sampling frequency can be determined from SFREQ which is described in Table 5-5. Thus the frame duration in seconds is:

$$\text{frame duration(seconds)} = (\text{NBLKS} + 1) \times 32/\text{Audio Sampling Frequency}$$

F.4.2.3 Frame Duration from Extension Substream Metadata

The parameters *nuRefClockCode* and *nuExSSFrameDurationCode* are used to determine the audio frame duration when an extension substream is present.

The frame duration is expressed by the number of clock cycles using the reference clock indicated by the value in *RefClockPeriod* (Table 7-3). The number of clock cycles is derived from *nuExSSFrameDurationCode* (clause 7.4.1) in the following manner:

$$\textit{frame duration(seconds)} = \textit{nuExSSFrameDurationCode} \times 512 \times \textit{RefClockPeriod}$$

F.4.3 Audio Access Unit Alignment in the PES packet

A valid sync word is aligned with the start of the PES packet data area. Valid DTS sync words are listed in Table F-1. Data_Alignment_Indicator in the PES packet header will indicate sync word alignment.

Table F-1: DTS-HD Sync Words

name	sync word	description
DTS_SYNCWORD_CORE	0x7ffe8001	core substream
DTS_SYNCWORD_SUBSTREAM	0x64582025	extension substream

When a core substream is present, DTS_SYNCWORD_CORE is aligned to the beginning of the PES payload. When only an extension substream is present, DTS_SYNCWORD_SUBSTREAM is aligned to the beginning of the PES payload.

A PES packet of DTS audio will contain at least one complete audio access unit. Multiple complete access units are permitted in a PES packet only when the Elementary Stream consists of a single substream.

The DTS core substream header parameter FSIZE, (clause 5.4.2), indicates the number of bytes in a core frame as FSIZE + 1 and the DTS-HD extension substream header parameter *nuExtSSFsize*, (clause 7.4.1), indicate the number of bytes in each extension substream frame (respectively), as shown in Figure F-1. The total PES packet payload of a single audio access unit will be the sum of these values for all substreams that are present.

If multiple substreams are present, the access units maintain an interleaved order of presentation, as illustrated in Figure F-1.

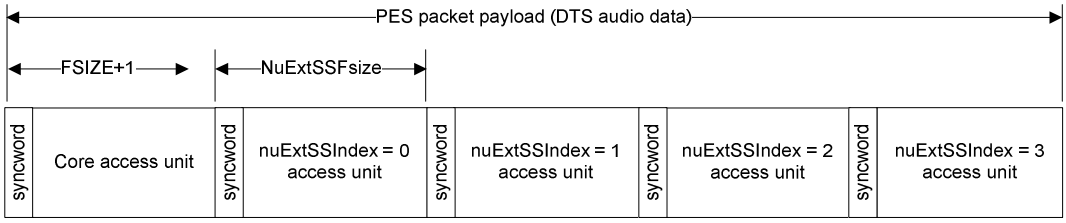


Figure F-1: PES packet payload

F.5 Implementation of DTS and DTS-HD Audio Stream Descriptors

The information needed to derive the elements within the audio descriptors can be derived from the respective elementary stream. DTS has tools available to implementers that will analyse DTS elementary streams and extract the information necessary to populate these parameters. DTS document #9302J81100 [i.1], describes the function calls and return structures. To obtain this tool and additional documentation, please direct all document requests to DTS Licensing at LicensingAdministration@dts.com.

Annex G (normative): DTS-HD Streaming with Using ISO/IEC 23009-1 (DASH)

G.1 Summary

This annex describes the Media Presentation Description (MPD) requirements for delivering DTS-HD® audio streams using MPEG DASH (Dynamic Adaptive Streaming over HTTP). In particular, the present document addresses the use of DTS-HD in audio adaptation sets, providing examples to support for the following DASH profiles:

- ISO Base media file format On Demand profile
- ISO Base media file format live profile
- Mpeg-2 TS main profile

G.2 MPEG DASH

G.2.1 Overview

Dynamic Adaptive Streaming over HTTP (DASH) [3] provides a standard-based adaptive media streaming model where chunks of media streams and file segments are requested with HTTP and spliced together by a client that controls the media delivery. DASH reuses widely deployed HTTP servers and caches for efficient delivery over existing content distribution infrastructure components such as Content Distribution Networks (CDNs), Network Address Translators (NATs) and firewalls. It provides a rich set of features to support on-demand, live streaming and time-shift applications and services to network-connected devices.

DASH is based on a hierarchical data model described by Media Presentation Description (MPD), which defines formats to announce resource identifiers for a collection of encoded and deliverable versions of media content. Media content is composed of single or multiple contiguous segments. The MPD provides sufficient information for a DASH client to provide a streaming service to the user by requesting segments from an HTTP web server and de-multiplexing, decoding and rendering the included media streams.

The segment formats specify the formats of the entity body of the HTTP response to an HTTP GET request or a partial HTTP GET with the indicated byte range using HTTP/1.1 to a resource identified in the MPD. DASH reuses the segment formats defined in ISO/IEC 14496-12 [5] and ISO/IEC 13818-1 [6]. Where values in the present document differ depending on the segment format used, the values to use are listed under the segment format name to which they correspond.

G.3 Media Presentation Description

G.3.1 Representation Base Type

Table G-1 is a summary of common attributes apply to Adaptation Set, Representation and Sub Representation elements.

Table G-1: Common attributes

Attribute	Description
@codecs	This attribute specifies the codecs used to encode all representations within the adaptation set and the value shall be one of "dtsc", "dtsh", "dtsl" or "dtse" corresponding to the composition of the elementary stream. **ISO/IEC 14496-12 [5]** This value may be derived from the coding name used in DTSSampleEntry box (Table E-2). **ISO/IEC 13818-1 [**6**]** For elementary streams signaled by the DTS audio descriptor, this value will always be 'dtsc'. For elementary streams signaled by the DTS-HD audio descriptor, asset_construction directly corresponds to StreamConstruction where the value can be looked up directly, (see ETSI EN 300 468 [1], Annex G)
@mimeType	**ISO/IEC 14496-12 [5]** For adaptation sets that conform to ISO/IEC 14496-12 [5], this value shall be set to either: "audio/mp4" for an ISO Base Media File that contains a DTS audio track but no accompanying video track "video/mp4" for an ISO Base Media File that contains a DTS audio track and one or more accompanying video tracks **ISO/IEC 13818-1 [**6**]** For adaptation sets that conform to ISO/IEC 13818-1 [6], this value shall be set to: "video/mp2t"
@audioSamplingRate	Sampling rate shall be equal to the maximum sampling frequency of the audio encoded in the DTS-HD bitstream. This value shall be a whole decimal number representing the sampling frequency in Hz. **ISO/IEC 14496-12 [5]** This value may be derived from DTSSamplingFrequency in the DTSSpecificBox (see Annex E). **ISO/IEC 13818-1 [6]** For elementary streams signaled by the DTS audio descriptor, the value may be derived from sample_rate_code in the descriptor (See ETSI EN 300 468 [1], Annex G). For elementary streams signaled by the DTS-HD audio descriptor, if an extension substream is present, this value may be derived from substream_0.sampling_frequency, otherwise the value may be derived from substream_core.sampling_frequency (see ETSI EN 300 468 [1], Annex G).

G.3.2 Audio Channel Configuration Descriptor

Audio channel configuration is used to identify the audio channel configuration scheme employed. Multiple AudioChannelConfiguration elements may be present, indicating that the Representation supports multiple audio channel configurations.

Table G-2: AudioChannelConfiguration attributes

Attribute	Description
@schemeIdUri	Scheme as described in "tag:dts.com,2014:dash:audio_channel_configuration:2012".
@value	AudioChannelConfiguration shall be set to the total number of discrete output channels represented in the stream, including LFE channels. The value shall be a whole decimal number in the range of 1 to 32. **ISO/IEC 14496-12 [5]** The channelcount parameter in DTSSampleEntry may be used to set AudioChannelConfiguration. (see Annex E). **ISO/IEC 13818-1 [6]** For elementary streams signaled by the DTS audio descriptor, this value may be derived from surround_mode, lfe_flag and extended_surround_flag, defined in the descriptor. For elementary streams signaled by the DTS-HD audio descriptor, if an extension substream is present, this value may be derived from substream_0.channel_count, otherwise the value9 may be derived from substream_core. channel_count (see ETSI EN 300 468 [1], Annex G).

G.3.3 Representation

A Representation describes a deliverable encoded version of one or more media content components. A DASH client may switch from Representation to Representation within an Adaptation Set to adapt to varying network bandwidth conditions. For DTS, bit rates may differ across the same DTS stream type in one Adaptation Set.

G.3.4 Coding Constraints

G.3.4.1 Coding Constraints for Seamless Stream Switching

Seamless stream switching shall enable a DASH client to switch from one DTS stream to another without interruption or muting between samples of the same encoded DTS audio content. To allow seamless stream switching between multiple DTS streams encoded with the same content within one Adaptation Set, the following parameters shall be constrained as follows:

- DTS audio coding name (@codecs) with the exception that for this purpose "dtsc" and "dtsh" may be considered as the same.
- Audio sampling frequency differences shall only be allowed as integer multiples (e.g. 48 000, 96 000, 192 000).

NOTE: The player needs to make note of the maximum audio sampling frequency that it intends to render and set its outputs accordingly, upsampling as necessary in order to facilitate seamless switching.

- Duration of synchronized frame.
- Audio channel arrangement.

G.3.4.2 Coding Constraints for Smooth Stream Switching

Smooth stream switching enables a DASH client to switch from one DTS stream to another by briefly fading out and then fading in ("V-fade") without a decoder reset. To allow smooth stream switching between multiple DTS streams encoded with the same content within an Adaptation Set, the following parameters shall be the same in all representations:

- DTS audio coding name (@codecs) with the exception that for this purpose "dtsc" and "dtsh" may be considered as the same.
- Audio sampling frequency differences shall only be allowed as integer multiples (e.g. 48 000, 96 000, 192 000). See the note regarding player behavior in clause G.3.4.1.
- Duration of synchronized frame.

G.3.4.3 Consideration for Switching of Audio Channel Arrangement (Informative)

Switching of audio channel arrangement can be achieved via smooth stream switching with V-fade. As frequent switching of audio channel arrangement may be disruptive, a DASH client application may employ intelligent algorithms to decide on switching of audio channel arrangement depending on user experience requirements and other factors. For example, based on hysteresis and improving bandwidth conditions, a DASH client application may return to a higher audio channel arrangement configuration after monitoring and adjusting media rates over a certain period of time.

G.4 Media Presentation Description Examples (Informative)

G.4.1 Example MPD for ISO Base media file format On Demand profile

The following is an example of a static presentation with self-initializing Media Segments and multiple base URLs. It describes the content available from two sources (cdn1 and cdn2) with four representations of the DTS-HD™ 5.1 audio provided at bitrates between 192 kbps and 510 kbps. The media presentation complies with the ISO base media file format On Demand profile, as defined in ISO/IEC 23009-1 [3].

```
<MPD
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="urn:mpeg:DASH:schema:MPD:2011"
    xsi:schemaLocation="urn:mpeg:DASH:schema:MPD:2011 DASH-MPD.xsd"
    type="static"
    mediaPresentationDuration="PT0H0M21.35S"
    minBufferTime="PT1.5S"
    profiles="urn:mpeg:dash:profile:isoff-on-demand:2011">
    <BaseURL>http://cdn1.example.com/</BaseURL>
    <BaseURL>http://cdn2.example.com/</BaseURL>
    <Period start="PT0S" duration="PT0H0M21.35S">
     <!—DTS-HD 5.1 channel English -->
        <AdaptationSet
            mimeType="audio/mp4"
            codecs="dtse"
            lang="en"
            audioSamplingRate="48000"
            startWithSAP="1"
            subsegmentStartsWithSAP="1" >
            <AudioChannelConfiguration
                schemeIdUri="tag:dts.com,2014:dash:audio_channel_configuration:2012"
                value="6"/>
            <ContentComponent id="100" contentType="audio"/>
            <Representation id="a1" bandwidth="192000">
                <BaseURL>dtse_192_dash.mp4</BaseURL>
                <SegmentBase indexRangeExact="true" indexRange="852-920"/>
            </Representation>
            <Representation id="a2" bandwidth="256000">
                <BaseURL>dtse_256_dash.mp4</BaseURL>
                <SegmentBase indexRangeExact="true" indexRange="853-921"/>
            </Representation>
            <Representation id="a3" bandwidth="384000">
                <BaseURL>dtse_384_dash.mp4</BaseURL>
                <SegmentBase indexRangeExact="true" indexRange="852-920"/>
            </Representation>
            <Representation id="a4" bandwidth="510000">
                <BaseURL>dtse_510_dash.mp4</BaseURL>
                <SegmentBase indexRangeExact="true" indexRange="853-921"/>
            </Representation>
        </AdaptationSet>
        <!-- Video -->
        <AdaptationSet
            mimeType="video/mp4"
            codecs="avc1.640028"
            lang="en"
            startWithSAP="1"
            subsegmentStartsWithSAP="1" >
```

```
            <ContentComponent id="101" contentType="video"/>
            <Representation id="v1" width="640" height="360" bandwidth="1020000">
                <BaseURL>avc_360p_dash.mp4</BaseURL>
                <SegmentBase indexRangeExact="true" indexRange="1176-1244"/>
            </Representation>
            <Representation id="v2" width="1280" height="720" bandwidth="5250000">
                <BaseURL>avc_720p_dash.mp4</BaseURL>
                <SegmentBase indexRangeExact="true" indexRange="1177-1245"/>
            </Representation>
            <Representation id="v3" width="1920" height="1080" bandwidth="8450000">
                <BaseURL>avc_1080p_dash.mp4</BaseURL>
                <SegmentBase indexRangeExact="true" indexRange="1178-1246"/>
            </Representation>
        </AdaptationSet>
    </Period>
</MPD>
```

Annex H (normative): DTS-HD Track Compliance with ISO/IEC 23000-19 (CMAF)

H.1 General guidelines for DTS-HD CMAF tracks

H.1.1 DTS-HD Conformance to Common Media Applications Format (CMAF)

This annex describes a media profile for DTS-HD to be used for CMAF applications. DTS-HD track files advertising CMAF compliance shall conform to annex G and this annex.

H.1.2 Codecs profiles and levels

DTS-HD streams and DTS-HD devices shall conform to the DTS-HD Basic profile described in Annex I.

Table E-2 provides a summary of the DTS-HD stream constructions that are possible. Note that some entries do not conform to the DTS-HD Basic profile and are therefore not included in the DTS-HD CMAF Media Profile, as indicated in the bottom row of Table H-1.

The maximum bit rates according to StreamConstruction type is shown in Table H-1.

Table H-1: Maximum Bitrates

Value of StreamConstruction	Maximum bitrate supported (Kits/sec)
1 to 4	1 536
5 to 13	6 144
14 to 16	24 576 (VBR)
18	768
17 to 19 – 21	Not permitted

H.1.3 Media access unit mapping to media samples

The SampleEntry for DTS-HD CMAF tracks shall conform to clause E.3. DTS-HD CMAF tracks shall be constrained to the following codingnames in the SampleEntry listed in Table H-2.

Table H-2: Valid codingname values for DTS-HD CMAF Tracks

codingname
dtsc
dtse
dtsh

H.1.4 Media access unit sequence mapping to CMAF fragments

A DTS-HD audio sample is described in clause E.2. Each DTS-HD audio sample is a sync sample.

H.1.5 CMAF track constraints for CMAF switching sets

DTS-HD CMAF tracks shall be constrained according to clause E.4. The restrictions apply to all audio samples within a track, and to tracks within a switching set where seamless switching is required.

H.1.6 CMAF media profile internet media type

DTS-HD CMAF track files delivered using DASH shall be signaled with the internet media type as indicated in Table G-1.

H.1.7 CMAF media profile brand

When the DTS-HD track conforms to clause H.2.2, the file type box ('ftyp') of a DTS-HD track may contain the brand 'dts1'. If the 'dts1' brand is present in the file type box, then the track shall comply with clause H.2.2.

H.2 Guidelines for DTS-HD CMAF media profiles

H.2.1 General

The frame duration of DTS-HD audio samples will be one of 512, 1 024, 2 048 or 4 096 decoded linear PCM samples per frame. In testing CMAF players, examples of each frame duration should be included. Table H-3 provides a summary of the recommended test vector properties to verify CMAF player compliance with the DTS-HD Media Profile. This represents the combination of bitrate, StreamConstruction and frame durations possible with the DTS-HD elementary stream.

Table H-3: Recommended Test Vector Parameters

StreamConstruction value	codingname	Audio frame duration	Max Bit rate (see note)
1	dtsc	512	1 536
1	dtsc	1 024	1 536
1	dtsc	2 048	768
5	dtsh	512	6 144
14	dtsh	512	24 576 (VBR)
18	dtse	4 096	768
NOTE: Maximum bit rates are nominal, expressed in kilo bits per second.			

H.2.2 Audio track format

A DTS-HD track conforming to the Media Profile defined by the 'dts1' brand shall conform to clause 10.2 of ISO/IEC 23000-19 [4], Annex E of the present document and one of 'dtsc', 'dtsh', or 'dtse' sample entries as defined in Table E-2, as constrained by clause H.1.2.

H.2.3 Loudness and dynamic range control

DTS-HD audio tracks should contain at least one set of DRC parameters, stored according to clause 5.8.3.

The DialNorm_rev2aux parameter stored in the Rev2Aux data block (described in clause 5.8.3) represents the loudness of the encoded content.

H.2.4 Audio parameters

No specific parameters are needed to initialize the DTS-HD decoder. Every sample is a sync sample, and contains all required metadata to begin decoding.

H.2.5 Audio presentation time adjustment

Delay can be compensated prior to stream encapsulation using the techniques specified in ISO/IEC 23000-19 [4], Annex G.5. The use of EditListBox is not required and further delay compensation in the receiver is not required.

H.3 Delivery Considerations for DTS-HD CMAF Tracks

The considerations for seamless switching and smooth switching for DASH, as described in Annex G, apply to the delivery of CMAF tracks.

H.4 Playback Considerations for DTS-HD CMAF Tracks

When playing back DTS-HD tracks in a supported MSE (HTML 5 Web Browser) environment, no special considerations are necessary for playback or segment boundaries.

Annex I (normative): DTS-HD Basic Profile

I.1 Overview

DTS-HD supports a number of bitstream options. This Annex describes a basic bitstream requirement and a minimum decoder profile requirement as a minimum interoperability requirement. This Annex describes the requirements of each.

Note that DTS-HD profiles defined for specific consumer and broadcast applications, and specific commercial implementations generally have additional requirements to this Basic Profile.

I.2 Basic Profile Decoder

All DTS-HD decoders and players shall support the following:

- Decoding of DTS core substreams
- Decoding of the DTS-HD Extension substream, including the following extensions
  - LBR
- Decoding of 5.1 channels
- Downmixing of multi-channel output to stereo
- Optional extensions not supported by a decoder shall be ignored and not interfere with playback as described in the above requirements

I.3 Basic Profile Bitstream

All DTS-HD bitstreams adhering to the Basic profile shall comply with the design rules of the present specification, and meet the following additional requirements:

- Inclusion of a core substream and/or an extension substream.
- If a core substream is not present, the extension substream shall contain an LBR extension.
- When a multi-channel DTS-HD bitstream contains 5 or more full bandwidth channels, and the bitstream includes a core, then the core shall contain 5 full bandwidth channels.
- When a DTS-HD bitstream includes a core, then a complete presentation shall be playable using only the core. Decoding of additional extensions shall only enhance the core, such as increasing bandwith, coding accuracy or adding output channels.
- In the case of a DTS-HD bitstream consisting of an LBR extension, the decoding and rendering beyond 5.1 channels shall be deemed optional.

Annex J (Informative): Other Registrations

J.1 Overview

This clause lists some select registrations that may be of interest to users of the DTS-HD codec.

J.2 MP4RA

Registrations related to DTS-HD that are listed in the MPEG-4 Registration Authority, which can be found at http://mp4ra.org are listed in this clause.

The following are listed as codecs with an assigned ObjectType:

dtsc

dtse

dtsh

dtsl

The following are listed as brands:

dts1

J.3 IANA

IANA registrations related to DTS-HD are listed in this clause.

The definition of vnd.dts can be found at https://www.iana.org/assignments/media-types/audio/vnd.dts

The definition of vnd.dts.hd can be found at https://www.iana.org/assignments/media-types/audio/vnd.dts.hd

History

Document history		
V1.1.1	August 2002	Publication
V1.2.1	December 2002	Publication
V1.3.1	August 2011	Publication
V1.4.1	September 2012	Publication
V1.5.1	May 2018	Publication
V1.6.1	August 2019	Publication