

INTERNATIONAL STANDARD

ISO/IEC 14496-3

Second edition
2001-12-15

AMENDMENT 1
2003-11-01

Information technology — Coding of audio-visual objects —

Part 3: **Audio**

AMENDMENT 1: Bandwidth extension

Technologies de l'information — Codage des objets audiovisuels —

Partie 3: Codage audio

AMENDEMENT 1: Extension de largeur de bande



Reference number
ISO/IEC 14496-3:2001/Amd.1:2003(E)

AVA Documents
Document Sharing Hub
© ISO/IEC 2003

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.

© ISO/IEC 2003

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Amendment 1 to ISO/IEC 14496-3:2001 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

Introduction

This document specifies the first Amendment to the ISO/IEC 14496-3:2001 standard. The document specifies the normative syntax of the SBR tool and the decoding process. An informative encoder description is given as well. Furthermore, this document specifies two new profiles, one based on the AAC LC Audio Object Type and one based on AAC in combination with SBR.

Information technology — Coding of audio-visual objects —

Part 3: Audio

AMENDMENT 1: Bandwidth extension

In ISO/IEC 14496-3:2001, Introduction, MPEG-4 general audio coding tools, add:

“

MPEG-4 SBR, (Spectral Band Replication) is a bandwidth extension tool used in combination with the AAC general audio codec. When integrated into the MPEG AAC codec, a significant improvement of the performance is available, which can be used to lower the bitrate or improve the audio quality. This is achieved by replicating the highband, i.e. the high frequency part of the spectrum. A small amount of data representing a parametric description of the highband is encoded and used in the decoding process. The data rate is by far below the data rate required when using conventional AAC coding of the highband.

“

Amendment Subpart 1

In Part 3: Audio, Subpart 1, in subclause 1.3 Terms and Definitions, add:

“

206. **SBR**: Spectral Band Replication.

”

and increase the index-number of subsequent entries.

In Part 3: Audio, Subpart 1, in subclause 1.5.1.1 Audio object type definition, replace table 1.1 with the following table:

Table 1.1 – Audio object definition

Tools/ Modules	Audio Object Type																																		
	gain control	block switching	window shapes - standard	window shapes – AAC LD	filterbank - standard	filterbank – SSR	TNS	LTP	intensity	coupling	MPEG-2 prediction	PNS	MS	SIAQ	FSS	upsampling filter tool	quantisation&coding - AAC	quantisation&coding – TwinVQ	quantisation&coding - BSAC	AAC ER Tools	ER payload syntax	EP Tool 1)	CELP	Silence Compression	HVXC	HVXC 4kbs VR	SA tools	SASBF	MIDI	HILN	TTSI	SBR	Remark	Object Type ID	
Null																																	0		
AAC main		X	X		X		X		X	X	X	X	X				X															2)	1		
AAC LC		X	X		X		X		X	X		X	X				X																	2	
AAC SSR	X	X	X			X	X		X	X		X	X				X																	3	
AAC LTP		X	X		X		X	X	X	X		X	X				X															2)	4		
SBR																																X	5		
AAC Scalable		X	X		X		X	X	X			X	X	X	X	X	X																6)	6	
TwinVQ		X	X		X		X	X					X					X																7	
CELP																							X											8	
HVXC																									X									9	
(Reserved)																																		10	
(Reserved)																																		11	
TTSI																															X			12	
Main synthetic																											X	X	X				3)	13	
Wavetable synthesis																												X	X				4)	14	
General MIDI																														X				15	
Algorithmic Synthesis and Audio FX																											X							16	
ER AAC LC		X	X		X		X		X			X	X				X			X	X	X												17	
(Reserved)																																		18	
ER AAC LTP		X	X		X		X	X	X			X	X				X			X	X	X											5)	19	
ER AAC scalable		X	X		X		X		X			X	X	X	X	X	X			X	X	X											6)	20	
ER TwinVQ		X	X		X		X						X					X			X	X												21	
ER BSAC		X	X		X		X		X			X	X						X		X	X												22	
ER AAC LD				X	X		X	X	X			X	X				X			X	X	X												23	
ER CELP																	X				X	X	X											24	
ER HVXC																					X	X			X	X								25	
ER HILN																					X	X									X			26	
ER Parametric																					X	X			X	X					X			27	
(Reserved)																																			28
(Reserved)																																			29
(Reserved)																																			30
(Reserved)																																			31

In Part 3: Audio, Subpart 1, in subclause 1.5.1.2 Description, add after 1.5.1.2.5, add:

“

1.5.1.2.6 SBR-Object

The SBR Object contains the SBR-Tool and can be combined with the audio object types indicated in Table 1.2A

Table 1.2A – Audio object types that can be combined with the SBR Tool

Audio Object Type	Combination with SBR Tool permitted	Object Type ID
Null		0
AAC main	X	1
AAC LC	X	2
AAC SSR	X	3
AAC LTP	X	4
SBR		5
AAC Scalable	X	6
TwinVQ		7
CELP		8
HVXC		9
(Reserved)		10
(Reserved)		11
TTSI		12
Main synthetic		13
Wavetable synthesis		14
General MIDI		15
Algorithmic Synthesis and Audio FX		16
ER AAC LC	X	17
(Reserved)		18
ER AAC LTP	X	19
ER AAC scalable	X	20
ER TwinVQ		21
ER BSAC		22
ER AAC LD		23
ER CELP		24
ER HVXC		25
ER HILN		26
ER Parametric		27
(Reserved)		28
(Reserved)		29
(Reserved)		30
(Reserved)		31

”

In Part 3: Audio, Subpart 1, subclause 1.5.2.1 (Profiles), replace:

“

Eight Audio Profiles have been defined:

”

with

“

Ten Audio Profiles have been defined:

”

and add after item 8:

“

9. The **AAC Profile** contains the audio object type 2 (AAC-LC).
10. The **High Efficiency AAC Profile** contains the audio object types 5 (SBR) and 2 (AAC LC) The High Efficiency AAC Profile is a superset of the AAC Profile

”

In Part 3: Audio, Subpart 1, replace Table 1.2 (Audio Profiles definition) with the following table:

“

Table 1.2 – Audio Profiles definition

Audio Object Type	Main Audio Profile	Scalable Audio Profile	Speech Audio Profile	Synthetic Audio Profile	High Quality Audio Profile	Low Delay Audio Profile	Natural Audio Profile	Mobile Audio Internet-working Profile	AAC Profile	High Efficiency AAC Profile	Object Type ID
Null											0
AAC main	X						X				1
AAC LC	X	X			X		X		X	X	2
AAC SSR	X						X				3
AAC LTP	X	X			X		X				4
SBR										X	5
AAC Scalable	X	X			X		X				6
TwinVQ	X	X					X				7
CELP	X	X	X		X	X	X				8
HVXC	X	X	X			X	X				9
(reserved)											10
(reserved)											11
TTSI	X	X	X	X		X	X				12
Main synthetic	X			X							13
Wavetable synthesis											14
General MIDI											15
Algorithmic Synthesis and Audio FX											16
ER AAC LC					X		X	X			17
(reserved)											18
ER AAC LTP					X		X				19
ER AAC Scalable					X		X	X			20
ER TwinVQ							X	X			21
ER BSAC							X	X			22
ER AAC LD						X	X	X			23
ER CELP					X	X	X				24
ER HVXC						X	X				25
ER HILN							X				26
ER Parametric							X				27
(reserved)											28
(reserved)											29
(reserved)											30
(reserved)											31

”

In Part 3: Audio, Subpart 1, subclause 1.5.2.2 (Complexity units), replace table 1.3 by the table below:

Table 1.3 – Complexity of Audio Object Types and SR conversion

Object Type	Parameters	PCU (MOPS)	RCU	Remarks
AAC Main	fs = 48 kHz	5	5	1)
AAC LC	fs = 48 kHz	3	3	1)
AAC SSR	fs = 48 kHz	4	3	1)
AAC LTP	fs = 48 kHz	4	4	1)
SBR	fs = 24/48 kHz (in/out) (SBR tool)	3	2.5	1)
	fs = 24/48 kHz (in/out) (Low Power SBR tool)	2	1.5	1)
	fs = 48/48 kHz (in/out) (Down Sampled SBR tool)	4.5	2.5	1)
	fs = 48/48 kHz (in/out) (Low Power Down Sampled SBR tool)	3	1.5	1)
AAC Scalable	fs = 48 kHz	5	4	1), 2)
TwinVQ	fs = 24 kHz	2	3	1)
CELP	fs = 8 kHz	1	1	
CELP	fs = 16 kHz	2	1	
CELP	fs = 8/16 kHz (bandwidth scalable)	3	1	
HVXC	fs = 8 kHz	2	1	
TTSI		-	-	4)
General MIDI		4	1	
Wavetable Synthesis	fs = 22.05 kHz	depends on bitstreams (3)	depends on bitstreams (3)	
Main Synthetic		depends on bitstreams (3)	depends on bitstreams (3)	
Algorithmic Synthesis and AudioFX		depends on bitstreams (3)	depends on bitstreams (3)	
Sampling Rate Conversion	rf = 2, 3, 4, 6	2	0.5	7)
ER AAC LC	fs = 48 kHz	3	3	1)
ER AAC LTP	fs = 48 kHz	4	4	1)
ER AAC Scalable	fs = 48 kHz	5	4	1), 2)
ER TwinVQ	fs = 24 kHz	2	3	1)
ER BSAC	fs = 48 kHz (input buffer size=26000bits)	4	4	1)
	fs = 48 kHz (input buffer size=106000bits)	4	8	
ER AAC LD	fs = 48 kHz	3	2	1)
ER CELP	fs = 8 kHz	2	1	
	fs = 16 kHz	3	1	
ER HVXC	fs = 8 kHz	2	1	
ER HILN	fs = 16 kHz, ns=93	15	2	6)
	fs = 16 kHz, ns=47	8	2	
ER Parametric	fs = 8 kHz, ns=47	4	2	5),6)

In Part 3: Audio, Subpart 1, subclause 1.5.2.3 (Levels within the profiles), add at the end:

“

- **Levels for the AAC Profile**

Table 1.7A - Levels for the AAC Profile

Level	Max. channels/ object	Max. sampling rate [kHz]	Max. PCU	Max. RCU
1	2	24	3	5
2	2	48	6	5
3	NA	NA	NA	NA
4	5	48	19	15
5	5	96	38	15

For the audio object type 2 (AAC LC), mono or stereo mixdown elements are not permitted.

The NA (Not Applicable) levels are introduced to emphasize the hierarchical structure of the AAC Profile and the High Efficiency AAC Profile. Hence, a decoder supporting the High Efficiency AAC Profile at a given level can decode an AAC Profile stream of the same or a lower level. The NA levels are not indicated in the audioProfileLevelIndication table (Table 1.7z).

- **Levels for the High Efficiency AAC Profile**

Table 1.8A - Levels for the High Efficiency AAC Profile

Level	Max. channels/ object	Max. AAC sampling rate, SBR not present [kHz]	Max. AAC sampling rate, SBR present [kHz]	Max. SBR sampling rate [kHz] (in/out)	Max. PCU	Max. RCU	Max. PCU Low power SBR	Max. RCU Low power SBR
1	NA	NA	NA	NA	NA	NA	NA	NA
2	2	48	24	24/48	9	10	7	8
3	2	48	48	48/48 (Note 1)	15	10	12	8
4	5	48	24/48 (Note 2)	48/48 (Note 1)	25	28	20	23
5	5	96	48	48/96	49	28	39	23

Note 1: For level 3 and level 4 decoders, it is mandatory to operate the SBR tool in downsampled mode if the sampling rate of the AAC core is higher than 24kHz. Hence, if the SBR tool operates on a 48kHz AAC signal, the internal sampling rate of the SBR tool will be 96kHz, however, the output signal will be downsampled by the SBR tool to 48kHz.

Note 2: For one or two channels the maximum AAC sampling rate, with SBR present, is 48kHz. For more than two channels the maximum AAC sampling rate, with SBR present, is 24kHz.

For the audio object type 2 (AAC LC), mono or stereo mixdown elements are not permitted.

”

In Part 3: Audio, Subpart 1, subclause 1.5.2.4 (Table 1.7z - audioProfileLevelIndication Values), replace the row:

“

0x28-0x7F	reserved for ISO use	-
-----------	----------------------	---

”

with

“

0x28	AAC Profile	L1
0x29	AAC Profile	L2
0x2A	AAC Profile	L4
0x2B	AAC Profile	L5
0x2C	High Efficiency AAC Profile	L2
0x2D	High Efficiency AAC Profile	L3
0x2E	High Efficiency AAC Profile	L4
0x2F	High Efficiency AAC Profile	L5
0x30-0x7F	reserved for ISO use	-

”

In Part 3: Audio, Subpart 1, in subclause 1.6.2.1 AudioSpecificConfig, replace table 1.8 with the following table:

“

Table 1.8 – Syntax of AudioSpecificConfig()

Syntax	No. of bits	Mnemonic
AudioSpecificConfig ()		
{		
audioObjectType;	5	uimsbf
samplingFrequencyIndex;	4	uimsbf
if (samplingFrequencyIndex==0xf)		
samplingFrequency;	24	uimsbf
channelConfiguration;	4	uimsbf
sbrPresentFlag = -1;		
if (audioObjectType == 5) {		
extensionAudioObjectType = audioObjectType;		
sbrPresentFlag = 1;		
extensionSamplingFrequencyIndex;	4	uimsbf
if (extensionSamplingFrequencyIndex==0xf)		
extensionSamplingFrequency;	24	uimsbf
audioObjectType;	5	uimsbf
}		
else {		
extensionAudioObjectType = 0;		
}		
if (audioObjectType == 1 audioObjectType == 2		
audioObjectType == 3 audioObjectType == 4		
audioObjectType == 6 audioObjectType == 7)		
GASpecificConfig();		
if (audioObjectType == 8)		
CelpSpecificConfig();		
if (audioObjectType == 9)		
HvxcSpecificConfig();		
if (audioObjectType == 12)		
TTSSpecificConfig();		
if (audioObjectType == 13 audioObjectType == 14		
audioObjectType == 15 audioObjectType==16)		
StructuredAudioSpecificConfig();		
/* the following Objects are Amendment 1 Objects */		
if (audioObjectType == 17 audioObjectType == 19		
audioObjectType == 20 audioObjectType == 21		
audioObjectType == 22 audioObjectType == 23)		
GASpecificConfig();		
if (audioObjectType == 24)		
ErrorResilientCelpSpecificConfig();		
if (audioObjectType == 25)		
ErrorResilientHvxcSpecificConfig();		
if (audioObjectType == 26 audioObjectType == 27)		
ParametricSpecificConfig();		
if (audioObjectType == 17 audioObjectType == 19		
audioObjectType == 20 audioObjectType == 21		
audioObjectType == 22 audioObjectType == 23		
audioObjectType == 24 audioObjectType == 25		
audioObjectType == 26 audioObjectType == 27) {		
epConfig;	2	uimsbf
if (epConfig == 2 epConfig == 3) {		
ErrorProtectionSpecificConfig();		
}		
if (epConfig == 3) {		

```

    directMapping;                                1          uimsbf
    if ( ! directMapping ) {
        /* tbd */
    }
}

if ( extensionAudioObjectType != 5 &&
    bits_to_decode() >= 16 ) {
    syncExtensionType;                            11         bsbf
    if (syncExtensionType == 0x2b7) {
        extensionAudioObjectType;                5          uimsbf
        if ( extensionAudioObjectType == 5 ) {
            sbrPresentFlag;                        1          uimsbf
            if (sbrPresentFlag == 1) {
                extensionSamplingFrequencyIndex;  4          uimsbf
                if ( extensionSamplingFrequencyIndex == 0xf )
                    extensionSamplingFrequency;  24         uimsbf
            }
        }
    }
}

```

In Part 3: Audio, Subpart 1, in subclause 1.6.2.2.1 Overview, replace table 1.9 by the following table:

“

Table 1.9 – Audio Object Types

Audio Object Type	Object Type ID	definition of elementary stream payloads and detailed syntax	Mapping of audio payloads to access units and elementary streams
AAC MAIN	1	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.2
AAC LC	2	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.2
AAC SSR	3	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.2
AAC LTP	4	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.2
SBR	5	ISO/IEC 14496-3 subpart 4	
AAC scalable	6	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.3
TwinVQ	7	ISO/IEC 14496-3 subpart 4	
CELP	8	ISO/IEC 14496-3 subpart 3	
HVXC	9	ISO/IEC 14496-3 subpart 2	
TTSI	12	ISO/IEC 14496-3 subpart 6	
Main synthetic	13	ISO/IEC 14496-3 subpart 5	
Wavetable synthesis	14	ISO/IEC 14496-3 subpart 5	
General MIDI	15	ISO/IEC 14496-3 subpart 5	
Algorithmic Synthesis and Audio FX	16	ISO/IEC 14496-3 subpart 5	
ER AAC LC	17	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.4
ER AAC LTP	19	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.4
ER AAC scalable	20	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.4
ER Twin VQ	21	ISO/IEC 14496-3 subpart 4	
ER BSAC	22	ISO/IEC 14496-3 subpart 4	
ER AAC LD	23	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.4
ER CELP	24	ISO/IEC 14496-3 subpart 3	
ER HVXC	25	ISO/IEC 14496-3 subpart 2	
ER HILN	26	ISO/IEC 14496-3 subpart 7	
ER Parametric	27	ISO/IEC 14496-3 subpart 2 and 7	

”

In Part 3: Audio, Subpart 1, under 1.6.3 Semantics, after 1.6.3.6 Direct Mapping add:

“

1.6.3.7 extensionSamplingFrequencyIndex

A four bit field indicating the output sampling frequency of the extension tool corresponding to the extensionAudioObjectType, according to Table 1.10.

1.6.3.8 extensionSamplingFrequency

The output sampling frequency of the extension tool corresponding to the extensionAudioObjectType. Either transmitted directly, or coded in the form of extensionSamplingFrequencyIndex.

1.6.3.9 bits_to_decode

A helper function; returns the number of bits not yet decoded in the current `AudioSpecificConfig()`, if the length of this element has been signaled by a system/transport layer. If the length of this element is unknown, `bits_to_decode()` returns 0.

1.6.3.10 syncExtensionType

Syncword which marks the beginning of appended extension configuration data. This configuration data corresponds to an extension tool of which the coded data is embedded (in a backward compatible manner) in that of the underlying `audioObjectType`. If `syncExtensionType` is present, the configuration data of the extension tool is separated from that of the underlying `audioObjectType`, which allows for backward compatible signaling (see subclause 1.6.5). Decoders that do not support the extension tool can ignore the extension tool configuration data. Note that this backward compatible signaling can only be used in MPEG-4 based systems that convey the length of the `AudioSpecificConfig()`.

1.6.3.11 sbrPresentFlag

A flag indicating the presence or absence of SBR data in case of `extensionAudioObjectType==5` (i.e. explicit SBR signaling, see subclause 1.6.5). The value `-1` indicates that the `sbrPresentFlag` was not conveyed in the `AudioSpecificConfig()`. In this case, a High Efficiency AAC Profile decoder shall be able to detect the presence of SBR data in the Elementary Stream (i.e. implicit SBR signaling, see subclause 1.6.5).

1.6.3.12 extensionAudioObjectType

A five bit field indicating the extension audio object type. This object type corresponds to an extension tool, which is used to enhance the underlying `audioObjectType`.

”

In Part 3: Audio, Subpart 1, after 1.6.4 Upstream, add the following subclause:

“

1.6.5 Signaling of SBR

1.6.5.1 Generating and Signaling AAC+SBR Content

The SBR tool in combination with the AAC coder provides a significant increase of audio compression efficiency. At the same time it allows for compatibility with existing AAC-only decoders. However, the audio quality for decoders without the SBR tool will of course be significantly lower than for those supporting the SBR tool. Therefore, depending on the application, a content provider or content creator will want to choose between the two alternatives given below. In general, the SBR data is always embedded in the AAC stream in a AAC compatible way (in the `extension_payload`), and SBR is a pure post processing step in the decoder. Therefore, compatibility can be achieved. However, by means of different signaling the content creator can select between the full-quality mode and the backward compatibility mode as follows.

1.6.5.1.1 Ensuring Full Audio Quality of AAC+SBR for the Listener

To ensure that all listeners get the full audio quality of AAC+SBR, the stream generated shall only play on SBR capable decoders (decoders that support the HE AAC Profile, hereinafter referred to as HE AAC Profile decoders). This is achieved by indicating the HE AAC profile and using the explicit, hierarchical signalling (signaling 2.A. as described below). As a result, decoders without SBR support will not play such streams. With regard to AAC-only streams, an HE AAC Profile decoders will decode all AAC Profile streams of the appropriate level, as the HE AAC Profile is a superset of the AAC Profile.

1.6.5.1.2 Achieving Backward Compatibility with Existing AAC-only Decoders

The aim of this mode is to get all AAC-based decoders to play the stream, even if they don't support the SBR tool. Compatible streams can be created using the following two signaling methods:

- a) indicating a profile containing AAC (e.g. the AAC Profile), except the HE AAC Profile, and using the explicit backward compatible signalling (2.B. as described below). This method is recommended for all MPEG-4 based systems in which the length of the AudioSpecificConfig() is known in the decoder. As this is not the case for LATM with audioMuxVersion==0 (see clause 1.7), this method cannot be used for LATM with audioMuxVersion==0. In explicit backward compatible signaling, SBR-specific configuration data is added at the end of the AudioSpecificConfig(). Decoders that do not know about SBR will ignore these parts, while HE AAC Profile decoders will detect its presence and configure the decoder accordingly.
- b) indicating a profile containing AAC (e.g. the AAC Profile, or an MPEG-2 AAC profile), except the HE AAC Profile, and using implicit signalling. In this mode, there is no explicit indication of the presence of SBR data. Instead, decoders check the presence while decoding the stream and use the SBR tool if SBR data is found. This is possible because SBR can be decoded without SBR-specific configuration data if a certain way of handling decoder output sample rate is obeyed, as described below for HE AAC Profile decoders.

Both methods lead to the result that the AAC part of an AAC+SBR streams will be decoded by AAC-only decoders. AAC+SBR decoders will detect the presence of SBR and decode the full quality AAC+SBR stream.

1.6.5.2 Implicit and Explicit Signaling of SBR

This subclause outlines the different signaling methods of SBR, and the decoder behavior for different types of signaling.

There are several ways to signal the presence of SBR data:

1. **implicit signaling:** If EXT_SBR_DATA or EXT_SBR_DATA_CRC extension_payload() elements are detected in the bitstream, this implicitly signals the presence of SBR data. The ability to detect and decode implicitly signaled SBR is mandatory for all High Efficiency AAC Profile (HE AAC Profile) decoders.
2. **explicit signaling:** The presence of SBR data is signaled explicitly by means of the SBR Audio Object Type in the AudioSpecificConfig(). When explicit signaling is used, implicit signaling shall not occur. Two different types of explicit signaling are available:
 - 2.A. **hierarchical signaling:** If the first audioObjectType (AOT) signaled is the SBR AOT, a second audio object type is signaled which indicates the underlying audio object type. This signaling method is not backward compatible.
 - 2.B. **backward compatible signaling:** The extensionAudioObjectType is signaled at the end of the AudioSpecificConfig(). This method shall only be used in systems that convey the length of the AudioSpecificConfig(). Hence, it shall not be used for LATM with audioMuxVersion==0.

Table 1.15A shows the decoder behavior depending on profile and audio object type indication when implicit or explicit signaling is used.

Table 1.15A – SBR Signaling and Corresponding Decoder Behavior

Bitstream characteristics				Decoder behavior (Note 4)	
Profile indication	extension AudioObjectType	sbrPresent Flag	raw_data_block	AAC decoders not supporting HE AAC Profile	AAC decoders supporting HE AAC Profile
Profiles with AAC support other than High Efficiency AAC Profile	!= SBR (signaling 1)	-1 (Note 1)	AAC	Play AAC	Play AAC
			AAC+SBR	Play AAC	Play at least AAC, should play AAC+SBR
	== SBR (signaling 2.B)	0 (Note 2)	AAC	Play AAC	Play AAC
		1 (Note 3)	AAC+SBR	Play AAC	Play at least AAC, should play AAC+SBR
High Efficiency AAC Profile	== SBR (signaling 2.A or 2.B)	1 (Note 3)	AAC+SBR	Unsupported Profile - Don't play	Play AAC+SBR
Note 1: Implicit signaling, check payload in order to determine output sampling frequency, or assume the presence of SBR data in the payload, giving an output sampling frequency of twice the sampling frequency indicated by samplingFrequency in the AudioSpecificConfig() (unless the down sampled SBR Tool is operated, or twice the sampling frequency indicated by samplingFrequency exceeds the maximum allowed output sampling frequency of the current level, in which case the output sampling frequency is the same as indicated by samplingFrequency). Note 2: Explicitly signals that there is no SBR data, hence no implicit signaling is present, and the output sampling frequency is given by samplingFrequency in the AudioSpecificConfig(). Note 3: Output sampling frequency is the extensionSamplingFrequency in AudioSpecificConfig(). Note 4: In all cases a decoder has to support the Profile and Level indicated in the bitstream in order to be able to decode and play the content of the bitstream.					

The upper part of Table 1.15A displays bitstream characteristics and decoder behavior if the profile indication is any profile with AAC, apart from the High Efficiency AAC Profile. The lower part displays bitstream characteristics and decoder behavior if the profile indication is the High Efficiency AAC Profile.

1.6.5.3 HE AAC Profile Decoder Behavior in Case of Implicit Signaling

If the presence of SBR data is backward compatible implicitly signaled (signaling 1, in the list above) the extensionAudioObjectType is not the SBR AOT, and the sbrPresentFlag is set to –1, indicating that implicit signaling may occur.

Since the HE AAC Profile decoder is a dual rate system, with the SBR Tool operating at twice the sample rate of the underlying AAC decoder, the output sample rate cannot be assumed to be that of the AAC decoder just because SBR is not explicitly signaled. The decoder shall determine the output sample rate by either of the following two methods:

- Check for the presence of SBR data in the bitstream prior to decoding. If no SBR data is found, the output sample rate is equal to that signaled as samplingFrequency in the AudioSpecificConfig(). If SBR data is found the output sample rate is twice that signaled as samplingFrequency in the AudioSpecificConfig

- Assume that the SBR data is available and decide the output sample rate to be twice that signaled in the `AudioSpecificConfig()`. If no SBR data is found once the decoding process has started, the SBR Tool can be used for upsampling only, as described in subclause 4.6.18.5.

The above only applies if twice the sample rate signaled in the `AudioSpecificConfig()` does not exceed the maximum output sample rate allowed for the current level. Hence, for a HE AAC Profile decoder of levels 2, 3, or 4, the output sample rate is equal to the sample rate signaled in the `AudioSpecificConfig()` if the latter exceeds 24kHz.

The down sampled SBR Tool shall be used when needed to ensure that the output sample rate does not exceed the maximum allowed sample rate of the present level of the High Efficiency AAC Profile decoder.

1.6.5.4 HE AAC Profile Decoder Behavior in Case of Explicit Signaling

If the presence of SBR data is explicitly signaled (signaling 2, in the list above) the presence of SBR data is backward compatible explicitly signaled (signaling 2.B) or non-backward compatible explicitly signaled (signaling 2.A).

For the backward compatible explicit signaled (signaling 2.B) the `extensionAudioObjectType` signaled is the SBR AOT. For this backward compatible explicit signaling the `sbrPresentFlag` is transmitted and can be either zero or one. If the `sbrPresentFlag` is zero, this indicates that SBR data is not present, and hence the HE AAC Profile decoder does not have to check the Fill-element for the presence of SBR data or make assumptions on the output sample rate in anticipation of SBR data. If the `sbrPresentFlag` is one, SBR data is present and the HE AAC Profile decoder shall operate the SBR Tool.

For the non-backward compatible explicit signaling of SBR (signaling 2.A) the `extensionAudioObjectType` signaled is the SBR AOT. For this hierarchical explicit signaling, the `sbrPresentFlag` is set to one if the `extensionAudioObjectType` is SBR. The `sbrPresentFlag` is not transmitted and hence it is not possible to explicitly signal the absence of implicit signaling. Hence, for the hierarchical explicit signaling, SBR data is always present and the HE AAC Profile decoder shall operate the SBR Tool.

The down sampled SBR Tool shall be operated if the output sample rate would otherwise exceed the maximum allowed output sample rate for the present level, or if the `extensionSamplingFrequency` is the same as the `samplingFrequency`.

”

In clause Annex 1.C (informative) Patent statements, replace the table by the following table:

“

- Legend:
1. The presence of a name of a company in the list below indicates that a patent statement has been received from that company
 2. The presence of a cross indicates that the statement identifies the part of the MPEG-4 version 2 standard to which the statement applies
 3. No cross in a line indicates that the statement does not identify which part of the standard the statement applies

	Company	Part 1	Part 2	Part 3	Part 5	Part 6
1.	Alcatel	x	x	x	x	x
2.	Apple	x			x	
3.	AT&T	x	x	x		
4.	BBC	x	x	x	x	x
5.	Bosch	x	x	x	x	x
6.	British Telecommunications	x	x	x	x	x
7.	Canon	x	x	x	x	x
8.	CCETT	x	x	x	x	x
9.	Coding Technologies			x		
10.	Columbia Innovation Enterprise	x	x	x	x	x
11.	Columbia University	x	x	x	x	x
12.	Creative	x		x	x	
13.	CSELT			x		
14.	DemoGraFX	x	x	x	x	x
15.	DirectTV	x	x	x		
16.	Dolby	x	x	x	x	x
17.	EPFL	x	x	x	x	
18.	ETRI	x	x	x	x	x
19.	France Telecom	x	x	x	x	x
20.	Fraunhofer	x	x	x	x	x
21.	Fujitsu	x	x	x	x	x
22.	GC Technology Corporation	x	x	x		
23.	General Instrument		x		x	
24.	Hitachi	x	x	x	x	x
25.	Hyundai	x	x	x	x	x
26.	IBM	x	x	x	x	x
27.	Institut fuer Rundfunktechnik	x	x	x		x
28.	Intertrust					
29.	JVC	x	x	x	x	x
30.	KDD Corporation	x	x			
31.	KPN	x	x	x	x	x
32.	LG Semicon					
33.	Lucent					
34.	Matsushita Electric Industrial Co., Ltd.	x	x	x	x	x
35.	Microsoft	x	x	x	x	x
36.	MIT					
37.	Mitsubishi	x	x	x	x	x

38.	Motorola		x		x	
39.	NEC	x	x	x	x	x
40.	NHK	x	x	x	x	x
41.	Nokia	x	x	x	x	
42.	NTT	x	x	x	x	x
43.	NTT Mobile Communication Networks			x		
44.	OKI	x	x	x	x	x
45.	Optibase	x			x	
46.	Philips	x	x	x	x	x
47.	PictureTel Corporation		x		x	
48.	Rockwell	x	x	x	x	x
49.	Samsung	x	x	x	x	
50.	Sarnoff	x	x	x	x	x
51.	Scientific Atlanta	x	x	x	x	x
52.	Sharp	x	x	x	x	x
53.	Siemens	x	x	x	x	x
54.	Sony	x	x	x	x	x
55.	Sun	x				
56.	Telenor	x	x	x	x	x
57.	Teltec DCU		x		x	
58.	Texas Instruments					
59.	Thomson	x	x	x	x	x
60.	Toshiba		x			
61.	Unisearch Ltd.		x		x	
62.	Vector Vision		x			

”

Amendment Subpart 4

In Part 3: Audio, Subpart 4, subclause 4.1.1 Technical Overview, 4.1.1.1 Encoder Decoder Block Diagrams, replace Figure 4.1 by the following figure:

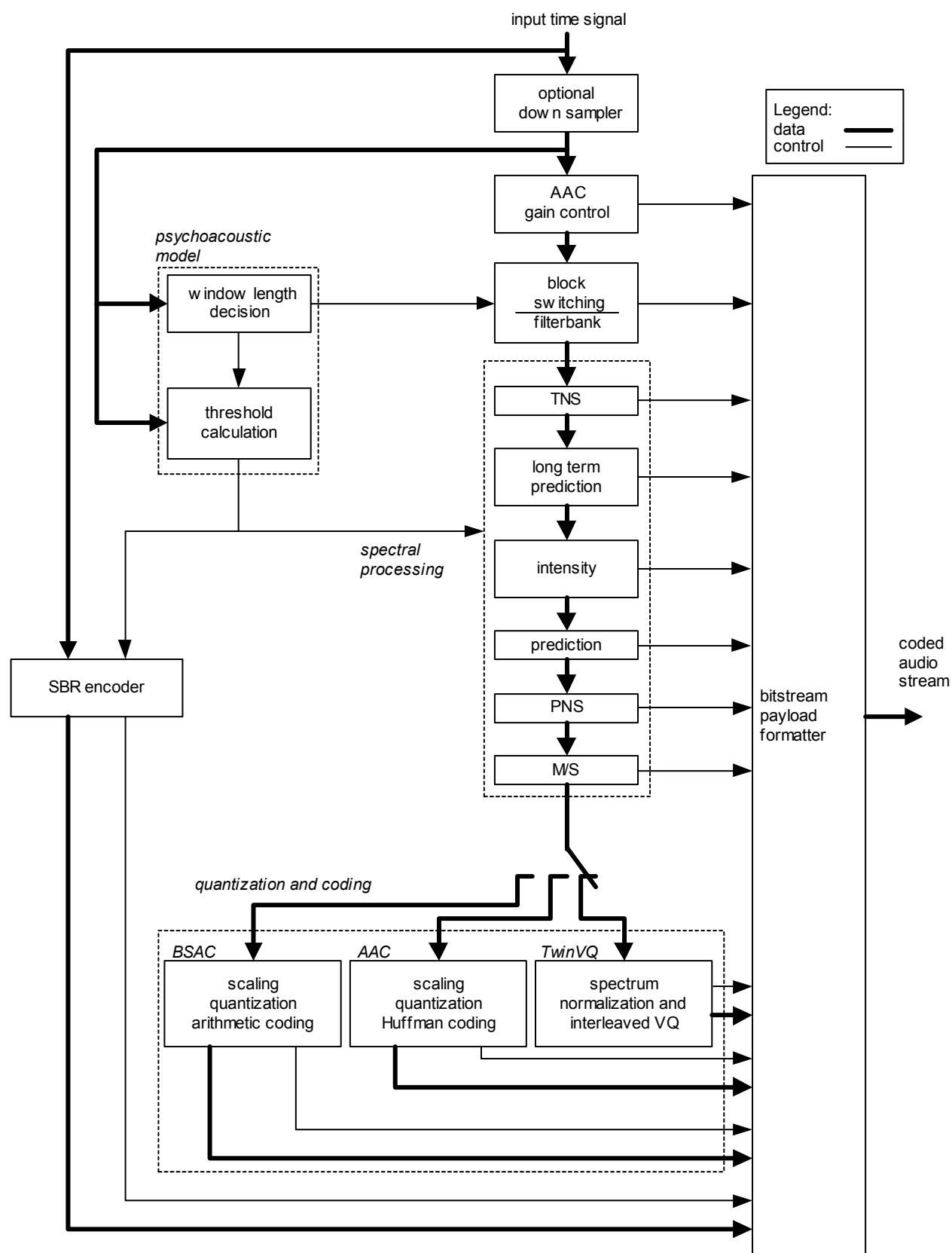


Figure 4.1 – Block diagram GA non scalable encoder

In Part 3: Audio, Subpart 4, subclause 4.1.1 Technical Overview, 4.1.1.1 Encoder Decoder Block Diagrams, add the following to Figure 4.2:

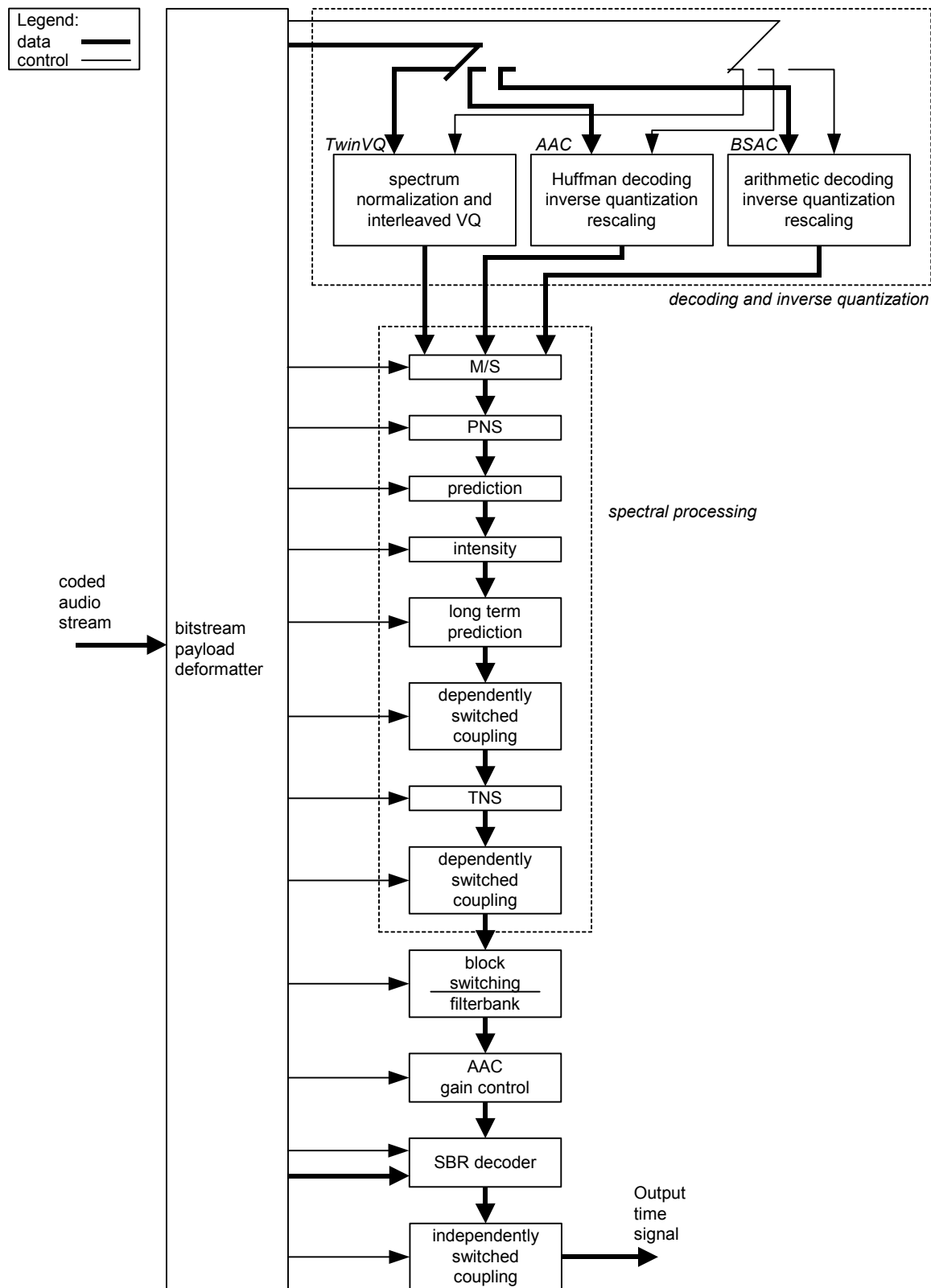


Figure 4.2 – Block diagram of the GA non scalable decoder

”

In Part 3: Audio, Subpart 4, subclause 4.1.1 Technical Overview, 4.1.1.2 Overview of the encoder and Decoder Tools, add the following:

“

The SBR tool regenerates the highband of the audio signal. It is based on replication of the sequences of harmonics, truncated during encoding. It adjusts the spectral envelope of the generated high-band and applies inverse filtering, and adds noise and sinusoidal components in order to recreate the spectral characteristics of the original signal.

The input to the SBR tool is:

- The quantized envelope data;
- Misc. control data;
- A time domain signal from the AAC core decoder.

The output of the SBR tool is:

- A time domain signal.

”

In Part 3: Audio, Subpart 4, Subclause 4.4.2.7 *Subsidiary payloads*, replace the definition of *extension_payload()*, Table 4.51:

“

Table 4.51 – Syntax of *extension_payload()*

Syntax	No. of bits	Mnemonic
<pre> extension_payload(cnt) { extension_type; align = 4; switch(extension_type) { case EXT_DYNAMIC_RANGE: return dynamic_range_info(); case EXT_SBR_DATA: return sbr_extension_data(id_aac, 0); case EXT_SBR_DATA_CRC: return sbr_extension_data(id_aac, 1); case EXT_FILL_DATA: fill_nibble; /* must be '0000' */ for (i=0; i<cnt-1; i++) { fill_byte[i]; /* must be '10100101' */ } return cnt; case EXT_DATA_ELEMENT: data_element_version; switch(data_element_version) { case ANC_DATA: loopCounter = 0; dataElementLength = 0; do { dataElementLengthPart; dataElementLength += dataElementLengthPart; loopCounter++; } while (dataElementLengthPart == 255); for (i=0; i<dataElementLength; i++) { data_element_byte[i]; } return (dataElementLength+loopCounter+1); case default: align = 0; } case EXT_FIL: case default: for (i=0; i<8*(cnt-1)+align; i++) { other_bits[i]; } return cnt; } } </pre>	4 4 4 8 8 8 1	uimsbf Note 1 Note 1 uimsbf uimsbf uimsbf uimsbf
Note 1: <i>id_aac</i> is the <i>id_syn_ele</i> of the corresponding AAC element (ID_SCE or ID_CPE) or ID_SCE in case of CCE.		

”

In Part 3: Audio, Subpart 4, after Subclause 4.4.2.7 *Subsidiary payloads*, add the following subclause:

“

4.4.2.8 Payloads for the audio object type SBR

Table 4.54A – Syntax of `sbr_extension_data()`

Syntax	No. of bits	Mnemonic
<pre> sbr_extension_data(id_aac, crc_flag) { num_sbr_bits = 0; if (crc_flag) { bs_sbr_crc_bits; num_sbr_bits += 10; } if (sbr_layer != SBR_STEREO_ENHANCE) { num_sbr_bits += 1; if (bs_header_flag) num_sbr_bits += sbr_header(); } num_sbr_bits += sbr_data(id_aac, bs_amp_res); num_align_bits = 8*cnt - 4 - num_sbr_bits; bs_fill_bits; return ((num_sbr_bits + num_align_bits + 4) / 8) } </pre>	10 1	uimsbf Note 1 Note 2 Note 2 uimsbf
Note 1: When the SBR tool is used with a non-scalable AAC core coder, the value of the helper variable <code>sbr_layer</code> is <code>SBR_NOT_SCALABLE</code>. When the SBR tool is used with a scalable AAC core coder, the value of the helper variable <code>sbr_layer</code> depends on the current layer and the scalability configuration of the AAC core coder as defined in Table 4.86 in Subclause 4.5.2.8.2.4. Note 2: <code>sbr_header()</code> and <code>sbr_data()</code> return the number of bits read (<code>cnt</code> is a parameter in <code>extension_payload()</code>).		

Table 4.55A – Syntax of sbr_header()

Syntax	No. of bits	Mnemonic
sbr_header() {		
bs_amp_res;	1	
bs_start_freq;	4	uimbsf, Note 1
bs_stop_freq;	4	uimbsf, Note 1
bs_xover_band;	3	uimbsf, Note 2
bs_reserved;	2	uimbsf
bs_header_extra_1;	1	
bs_header_extra_2;	1	
if (bs_header_extra_1) {		Note 3
bs_freq_scale;	2	uimbsf
bs_alter_scale;	1	
bs_noise_bands;	2	uimbsf
}		
if (bs_header_extra_2) {		Note 3
bs_limiter_bands;	2	uimbsf
bs_limiter_gains;	2	uimbsf
bs_interpol_freq;	1	
bs_smoothing_mode;	1	
}		
}		
Note 1: bs_start_freq and bs_stop_freq shall define a frequency band that does not exceed the limits defined in subclause 4.6.18.3.6.		
Note 2: Index to the master frequency band table, indicating where the current SBR range begins		
Note 3: If this bit is not set the default values for the underlying bitstream elements should be used.		

Table 4.56A – Syntax of sbr_data()

Syntax	No. of bits	Mnemonic
<pre> sbr_data(id_aac, bs_amp_res) { switch (sbr_layer) { case SBR_NOT_SCALABLE switch (id_aac) { case ID_SCE sbr_single_channel_element(bs_amp_res) break; case ID_CPE sbr_channel_pair_element(bs_amp_res) break; } break; case SBR_MONO_BASE sbr_channel_pair_base_element(bs_amp_res) break; case SBR_STEREO_ENHANCE sbr_channel_pair_enhance_element(bs_amp_res) break; case SBR_STEREO_BASE sbr_channel_pair_element(bs_amp_res) break; } } </pre>		Note 1
Note 1: When the SBR tool is used with a non-scalable AAC core coder, the value of the helper variable sbr_layer is SBR_NOT_SCALABLE. When the SBR tool is used with a scalable AAC core coder, the value of the helper variable sbr_layer depends on the current layer and the scalability configuration of the AAC core coder as defined in Table 4.86 in Subclause 4.5.2.8.2.4.		

Table 4.57A – Syntax of sbr_single_channel_element()

Syntax	No. of bits	Mnemonic
sbr_single_channel_element(bs_amp_res)		
{		
if (bs_data_extra)	1	
bs_reserved;	4	uimsbf
sbr_grid(0);		
sbr_dtdf(0);		
sbr_invf(0);		
sbr_envelope(0, 0, bs_amp_res);		
sbr_noise(0, 0);		
if (bs_add_harmonic_flag[0])	1	
sbr_sinusoidal_coding(0);		
if (bs_extended_data) {	1	
cnt = bs_extension_size;	4	uimsbf
if (cnt == 15)		
cnt += bs_esc_count;	8	uimsbf
num_bits_left = 8 * cnt;		
while (num_bits_left > 7) {		
bs_extension_id;	2	uimsbf
num_bits_left -= 2;		
sbr_extension(bs_extension_id, num_bits_left);		Note 1
}		
}		
}		
Note 1: sbr_extension() shall decrease the variable num_bits_left by the number of bits read from the bitstream within sbr_extension(). The sbr_extension() element is reserved for future use.		

Table 4.58A – Syntax of sbr_channel_pair_element()

Syntax	No. of bits	Mnemonic
sbr_channel_pair_element(bs_amp_res)		
{		
if (bs_data_extra) {	1	
bs_reserved ;	4	uimsbf
bs_reserved ;	4	uimsbf
}		
if (bs_coupling) {	1	
sbr_grid(0);		
sbr_dtdf(0);		
sbr_dtdf(1);		
sbr_invf(0);		
sbr_envelope(0,1, bs_amp_res);		
sbr_noise(0,1);		
sbr_envelope(1,1, bs_amp_res);		
sbr_noise(1,1);		
} else {		
sbr_grid(0);		
sbr_grid(1);		
sbr_dtdf(0);		
sbr_dtdf(1);		
sbr_invf(0);		
sbr_invf(1);		
sbr_envelope(0,0, bs_amp_res);		
sbr_envelope(1,0, bs_amp_res);		
sbr_noise(0,0);		
sbr_noise(1,0);		
}		
if (bs_add_harmonic_flag [0])	1	
sbr_sinusoidal_coding(0);		
if (bs_add_harmonic_flag [1])	1	
sbr_sinusoidal_coding(1);		
if (bs_extended_data) {	1	
cnt = bs_extension_size ;	4	uimsbf
if (cnt == 15)		
cnt += bs_esc_count ;	8	uimsbf
num_bits_left = 8 * cnt;		
while (num_bits_left > 7) {		
bs_extension_id ;	2	uimsbf
num_bits_left -= 2;		
sbr_extension(bs_extension_id, num_bits_left);		Note 1
}		
}		
}		
Note 1: sbr_extension() shall decrease the variable num_bits_left by the number of bits read from the bitstream within sbr_extension(). The sbr_extension() element is reserved for future use.		

Table 4.59A – Syntax of sbr_channel_pair_base_element()

Syntax	No. of bits	Mnemonic
sbr_channel_pair_base_element(bs_amp_res)		
{		
if (bs_data_extra) {	1	
bs_reserved;	4	uimsbf
bs_reserved;	4	uimsbf
}		
bs_coupling	1	Note 1
sbr_grid(0);		
sbr_dtdf(0);		
sbr_invf(0);		
sbr_envelope(0,1, bs_amp_res);		
sbr_noise(0,1);		
if (bs_add_harmonic_flag[0])	1	
sbr_sinusoidal_coding(0);		
if (bs_extended_data) {	1	
cnt = bs_extension_size;	4	uimsbf
if (cnt == 15)		
cnt += bs_esc_count;	8	uimsbf
num_bits_left = 8 * cnt;		
while (num_bits_left > 7) {		
bs_extension_id;	2	uimsbf
num_bits_left -= 2;		
sbr_extension(bs_extension_id, num_bits_left);		Note 2
}		
}		
}		
Note 1: bs_coupling shall have the value 1.		
Note 2: sbr_extension() shall decrease the variable num_bits_left by the number of bits read from the bitstream within sbr_extension(). The sbr_extension() element is reserved for future use.		

Table 4.60A – Syntax of sbr_channel_pair_enhance_element()

Syntax	No. of bits	Mnemonic
sbr_channel_pair_enhance_element(bs_amp_res)		
{		
sbr_dtdf(1);		
sbr_envelope(1,1, bs_amp_res);		
sbr_noise(1,1);		
if (bs_add_harmonic_flag[1])	1	
sbr_sinusoidal_coding(1);		
}		

Table 4.61A – Syntax of `sbr_grid()`

Syntax	No. of bits	Mnemonic
<code>sbr_grid(ch)</code>		
{		
switch (bs_frame_class) {	2	uimsbf
case FIXFIX		
bs_num_env [ch] = 2 ^{tmp} ;	2	uimsbf , Note 1
if (bs_num_env [ch] == 1)		
bs_amp_res = 0;		
bs_freq_res [ch][0];	1	
for (env = 1; env < bs_num_env [ch]; env++)		
bs_freq_res [ch][env] = bs_freq_res [ch][0];		
break;		
case FIXVAR		
bs_var_bord_1 [ch];	2	uimsbf
bs_num_env [ch] = bs_num_rel_1 [ch] + 1;	2	uimsbf
for (rel = 0; rel < bs_num_env [ch]-1; rel++)		
bs_rel_bord_1 [ch][rel] = 2* tmp + 2;	2	uimsbf
ptr_bits = ceil (log (bs_num_env [ch] + 1) / log (2));		Note 2
bs_pointer [ch];	ptr_bits	uimsbf
for (env = 0; env < bs_num_env [ch]; env++)		
bs_freq_res [ch][bs_num_env [ch] - 1 - env];	1	
break;		
case VARFIX		
bs_var_bord_0 [ch];	2	uimsbf
bs_num_env [ch] = bs_num_rel_0 [ch] + 1;	2	uimsbf
for (rel = 0; rel < bs_num_env [ch]-1; rel++)		
bs_rel_bord_0 [ch][rel] = 2* tmp + 2;	2	uimsbf
ptr_bits = ceil (log (bs_num_env [ch] + 1) / log (2));		Note 2
bs_pointer [ch];	ptr_bits	uimsbf
for (env = 0; env < bs_num_env [ch]; env++)		
bs_freq_res [ch][env];	1	
break;		
case VARVAR		
bs_var_bord_0 [ch];	2	uimsbf
bs_var_bord_1 [ch];	2	uimsbf
bs_num_rel_0 [ch];	2	uimsbf
bs_num_rel_1 [ch];	2	uimsbf
bs_num_env [ch] = bs_num_rel_0 [ch] + bs_num_rel_1 [ch] + 1;		Note 1
for (rel = 0; rel < bs_num_rel_0 [ch]; rel++)		
bs_rel_bord_0 [ch][rel] = 2* tmp + 2;	2	uimsbf
for (rel = 0; rel < bs_num_rel_1 [ch]; rel++)		
bs_rel_bord_1 [ch][rel] = 2* tmp + 2;	2	uimsbf
ptr_bits = ceil (log(bs_num_env [ch] + 1) / log (2));		Note 2
bs_pointer [ch];	ptr_bits	uimsbf
for (env = 0; env < bs_num_env [ch]; env++)		
bs_freq_res [ch][env];	1	
break;		
}		
if (bs_num_env [ch] > 1)		
bs_num_noise [ch] = 2;		
else		
bs_num_noise [ch] = 1;		
}		
Note 1: bs_num_env is restricted according to subclause 4.6.18.3.6.		
Note 2: the division (/) is a float division without rounding or truncation.		

Table 4.62A – Syntax of sbr_dtdf()

Syntax	No. of bits	Mnemonic
sbr_dtdf(ch)		
{		
for (env = 0; env < bs_num_env[ch]; env++)		
bs_df_env [ch][env];	1	
for (noise = 0; noise < bs_num_noise[ch]; noise++)		
bs_df_noise [ch][noise];	1	
}		

Table 4.63A – Syntax of sbr_invf()

Syntax	No. of bits	Mnemonic
sbr_invf(ch)		
{		
for (n = 0; n < num_noise_bands[ch]; n++)		
bs_invf_mode [ch][n];	2	Note 1 uimsbf
}		
Note 1: num_noise_bands[ch] is derived from the header, according to subclause 4.6.18.3 and is named N _Q .		

Table 4.64A – Syntax of sbr_envelope()

Syntax	No. of bits	Mnemonic
<pre> sbr_envelope(ch, bs_coupling, bs_amp_res) { if (bs_coupling) { if (ch) { if (bs_amp_res) { t_huff = t_huffman_env_bal_3_0dB; f_huff = f_huffman_env_bal_3_0dB; } else { t_huff = t_huffman_env_bal_1_5dB; f_huff = f_huffman_env_bal_1_5dB; } } else { if (bs_amp_res) { t_huff = t_huffman_env_3_0dB; f_huff = f_huffman_env_3_0dB; } else { t_huff = t_huffman_env_1_5dB; f_huff = f_huffman_env_1_5dB; } } } else { if (bs_amp_res) { t_huff = t_huffman_env_3_0dB; f_huff = f_huffman_env_3_0dB; } else { t_huff = t_huffman_env_1_5dB; f_huff = f_huffman_env_1_5dB; } } } for (env = 0; env < bs_num_env[ch]; env++) { if (bs_df_env[ch][env] == 0) { if (bs_coupling && ch) { if (bs_amp_res) bs_data_env[ch][env][0] = bs_env_start_value_balance; else bs_data_env[ch][env][0] = bs_env_start_value_balance; } else { if (bs_amp_res) bs_data_env[ch][env][0] = bs_env_start_value_level; else bs_data_env[ch][env][0] = bs_env_start_value_level; } for (band = 1; band < num_env_bands[bs_freq_res[ch][env]]; band++) bs_data_env[ch][env][band] = sbr_huff_dec(f_huff, bs_codeword); } else { for (band = 0; band < num_env_bands[bs_freq_res[ch][env]]; band++) bs_data_env[ch][env][band] = sbr_huff_dec(t_huff, bs_codeword); } } </pre>	5 6 6 7 1..18 1..18	uimsbf uimsbf uimsbf uimsbf Note 1 Note 2 Note 1 Note 2
Note 1: num_env_bands[bs_freq_res[ch][env]] is derived from the header according to subclause 4.6.18.3 and is named n. Note 2: sbr_huff_dec() is defined in Annex 4.A.6.1.		

Table 4.65A – Syntax of sbr_noise()

Syntax	No. of bits	Mnemonic
<pre> sbr_noise(ch,bs_coupling) { if (bs_coupling) { if (ch) { t_huff = t_huffman_noise_bal_3_0dB; f_huff = f_huffman_noise_bal_3_0dB; } else { t_huff = t_huffman_noise_3_0dB; f_huff = f_huffman_noise_3_0dB; } } else { t_huff = t_huffman_noise_3_0dB; f_huff = f_huffman_noise_3_0dB; } for (noise = 0; noise < bs_num_noise[ch]; noise++) { if (bs_df_noise[ch][noise] == 0) { if (bs_coupling && ch) bs_data_noise[ch][noise][0] = bs_noise_start_value_balance; else bs_data_noise[ch][noise][0] = bs_noise_start_value_level; for (band = 1; band < num_noise_bands[ch]; band++) bs_data_noise[ch][noise][band] = sbr_huff_dec(f_huff,bs_codeword); } else { for (band = 0; band < num_noise_bands[ch]; band++) bs_data_noise[ch][noise][band] = sbr_huff_dec(t_huff,bs_codeword); } } } </pre>	5 5 1..18 1..18	uimsbf uimsbf Note 1 Note 2 Note 1 Note 2
Note 1: num_noise_bands[ch] is derived from the header according to subclause 4.6.18.3 and is named N_Q. Note 2: sbr_huff_dec() is defined in Annex 4.A.6.1.		

Table 4.66A – Syntax of sbr_sinusoidal_coding()

Syntax	No. of bits	Mnemonic
<pre> sbr_sinusoidal_coding(ch) { for (n = 0; n < num_high_res[ch]; n++) bs_add_harmonic[ch][n] } </pre>	1	Note 1
Note 1: num_high_res[ch] is derived from the header according to subclause 4.6.18.3 and is named N_{High}.		

”

In Part 3: Audio, Subpart 4, Subclause 4.5.2.1.6 Fill element (FIL), replace defined values of the extension_type field in Table 4.59:

“

Table 4.59 – Values of the extension_type field

Symbol	Value of extension_type	Purpose
EXT_FILL	'0000'	bitstream filler
EXT_FILL_DATA	'0001'	bitstream data as filler
EXT_DATA_ELEMENT	'0010'	data element
EXT_DYNAMIC_RANGE	'1011'	dynamic range control
EXT_SBR_DATA	'1101'	SBR enhancement
EXT_SBR_DATA_CRC	'1110'	SBR enhancement with CRC
-	all other values	reserved

”

And add the following restriction:

“

Fill elements containing an extension_payload with an extension_type of EXT_SBR_DATA or EXT_SBR_DATA_CRC shall not contain any other extension_payload of any other extension_type. For fill elements containing an extension_payload with an extension_type of EXT_SBR_DATA or EXT_SBR_DATA_CRC, the fill_element_count field shall be set equal to the total length in bytes, including the SBR enhancement data plus the extension_type field.

”

In Part 3: Audio, Subpart 4, subclause 4.5.2.4.1 (Error sensitivity category assignment), replace table 4.64 with:

“

Table 4.64 – Error sensitivity category assignment

category	payload	mandatory	leads / may lead to one instance per	Description
0	main	yes	CPE	commonly used side information
1	main	yes	ICS	channel dependent side information
2	main	no	ICS	error resilient scale factor data
3	main	no	ICS	TNS data
4	main	yes	ICS	spectral data
5	extended	no	EPL	extension type / data_element_version
6	extended	no	EPL	DRC data
7	extended	no	EPL	bit stuffing
8	extended	no	EPL	ANC data
9	extended	no	EPL	SBR data

”

In Part 3: Audio, Subpart 4, subclause 4.5.2 Decoding of the GA bitstream payloads, add the following subclause 4.5.2.8 Payloads for the audio object type SBR:

“

4.5.2.8 Payloads for the audio object type SBR

4.5.2.8.1 Definitions

bs_reserved	Bits reserved for future use, default value is zero.
sbr_extension_data()	Syntactic element that contains the SBR extension data.
bs_sbr_crc_bits	Cyclic redundancy checksum for the SBR extension data. The CRC code is defined by the generator polynomial $G_{10}(x) = x^{10} + x^9 + x^5 + x^4 + x + 1$ and the initial value for the CRC calculation is zero.
bs_header_flag	Indicates if an SBR header is present.
sbr_header()	Syntactic element that contains the SBR header.
sbr_data()	Syntactic element that contains the SBR data.
bs_fill_bits	Byte alignment bits.
bs_amp_res	Defines the resolution of the envelope estimates according to:

Table 4.70A – bs_amp_res

bs_amp_res	Meaning
0	1.5 dB
1	3.0 dB

bs_start_freq	Input parameter to function that calculates start of master frequency table.
bs_stop_freq	Input parameter to function that calculates stop of master frequency table.
bs_xover_band	Index to master frequency table.
bs_header_extra_1	Indicates if the optional header part 1 is present.
bs_header_extra_2	Indicates if the optional header part 2 is present.
bs_freq_scale	Input parameter to function that calculates master frequency table, defined by:

Table 4.71A – bs_freq_scale

bs_freq_scale	Meaning
0	linear
1	12 bands/octave
2 (default)	10 bands/octave
3	8 bands/octave

bs_alter_scale	Input parameter to function that calculates master frequency table, defined by:
-----------------------	---

Table 4.72A – bs_alter_scale

bs_alter_scale	Meaning for bs_freq_scale = 0	Meaning for bs_freq_scale > 0
0	no grouping of channels	no alteration
1 (default)	groups of 2 channels	extra wide bands in highest range

bs_noise_bands	Input parameter to function that calculates noise band table, defined by:
-----------------------	---

Table 4.73A – bs_noise_bands

bs_noise_bands	Meaning
0	1 band
1	1 band/octave
2 (default)	2 bands/octave
3	3 bands/octave

bs_limiter_bands Input parameter to function that calculates limiter band table, defined by:

Table 4.74 – bs_limiter_bands

bs_limiter_bands	Meaning	Note
0	1 band	single band
1	1.2 bands/octave	multi-band
2 (default)	2.0 bands/octave	multi-band
3	3.0 bands/octave	multi-band

bs_limiter_gains Defines the maximum gain of the limiters according to:

Table 4.75A – bs_limiter_gains

bs_limiter_gains	Meaning
0	-3 dB Max Gain
1	0 dB Max Gain
2 (default)	3 dB Max Gain
3	inf. dB Max Gain(i.e. limiter off)

bs_interpol_freq Defines if the frequency interpolation shall be applied according to:

Table 4.76A – bs_interpol_freq

bs_interpol_freq	Meaning
0	off
1 (default)	on

bs_smoothing_mode Defines if smoothing shall be applied according to:

Table 4.77A – bs_smoothing_mode

bs_smoothing_mode	Meaning
0	on
1 (default)	off

sbr_single_channel_element() Syntactic element that contains data for an SBR single channel element.

sbr_channel_pair_element() Syntactic element that contains data for an SBR channel pair element.

sbr_channel_pair_base_element() Syntactic element that contains base-layer data for an SBR channel pair element, in a scalable system.

sbr_channel_pair_enhance_element() Syntactic element that contains enhancement-layer data for an SBR channel pair element, in a scalable system.

bs_data_extra Indicates if reserved bits in the SBR data part are present.

sbr_grid() Syntactic element that contains the time frequency grid.

sbr_dtdf() Syntactic element that contains the information on how the envelope and noise data is delta coded.

sbr_invf() Syntactic element that contains the inverse filtering data.

sbr_envelope() Syntactic element that contains the huffman coded envelope data.

sbr_noise() Syntactic element that contains the huffman coded noise floor data.

bs_add_harmonic_flag Indicates if sinusoidal coding information is present.

sbr_sinusoidal_coding() Syntactic element that contains sinusoidal coding data.

bs_extended_data Indicates if an SBR extended data element is present.

bs_extension_size Defines the size of the SBR extended data element in bytes.

bs_esc_count Further defines the size of the SBR extended data element in cases where the size is larger than 14 bytes.