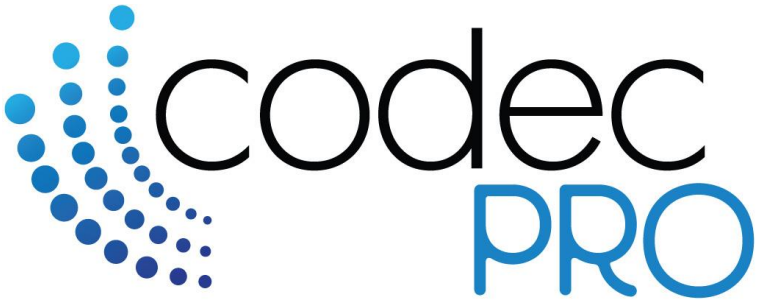

EVS Codec

Universal API Implementation

Technical Documentation

Version 10.1 Revision A
November 2024

CodecPro Incorporated
2162 Laurier East
Montreal, QC H2H 1C2 CANADA
sales@codecpro.com
www.codecpro.com

Contents

Revision history

November 2024	Revised API description for RTP payload packing and Table 1 - Speech frame type to size
October 2024	Added API description for RTP payload packing
May 2024	Fixed the table of contents.
May 2023	Added Intel platforms in the Complexity section.
January 2022	Parameter bitstreamformat was added to evs_decoderOption.
March 2021	Updated Complexity section and added the Memory usage section.
August 2019	hScratch parameter descriptions updated.
December 2016	Cosmetic modifications.
March 2016	First release of this document.

References

[1] TS 26.441 - Codec for Enhanced Voice Services (EVS); General overview
https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=1463

[2] TS 26.442 - Codec for Enhanced Voice Services (EVS); ANSI C code (fixed-point)
https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=1464

[3] TS 26.443 - Codec for Enhanced Voice Services (EVS); ANSI C code (floating-point)
https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=1465

[4] TS 26.452 - Codec for Enhanced Voice Services (EVS); ANSI C code (alternative fixed-point)
https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3473

[5] TS 26.444 - Codec for Enhanced Voice Services (EVS); Test sequences
https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=1466

[6] TS 26.445 - Codec for Enhanced Voice Services (EVS); Detailed algorithmic description
https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=1467

[7] TS 26.446 - Codec for Enhanced Voice Services (EVS); Adaptive Multi-Rate - Wideband (AMR-WB) backward compatible functions
https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=1468

[8] TS 26.447 - Codec for Enhanced Voice Services (EVS); Error concealment of lost packets
https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=1469

[9] TS 26.448 - Codec for Enhanced Voice Services (EVS); Jitter buffer management
https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=1470

[10] TS 26.449 - Codec for Enhanced Voice Services (EVS); Comfort Noise Generation (CNG) aspects
https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=1471

[11] TS 26.450 - Codec for Enhanced Voice Services (EVS); Discontinuous Transmission (DTX)
https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=1472

[12] TS 26.451 - Codec for Enhanced Voice Services (EVS); Voice Activity Detection (VAD)
https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=1473

[13] TS 26.114 - IP Multimedia Subsystem (IMS); Multimedia Telephony; Media handling and interaction.
https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=1404

EVS Codec

The Enhanced Voice Services (EVS) coder consists of the multi-rate audio coder optimized for operation with voice and music/mixed content signals, a source-controlled rate scheme including a voice/sound activity detector and a comfort noise generation system, and an error concealment mechanism to combat the effects of transmission errors and lost packets. Jitter buffer management for VoIP communication is also provided.

Four PCM sampling rates are supported; 8 000, 16 000, 32 000 and 48 000 samples/s and the bit rates for the encoded bit stream may be 5.9, 7.2, 8.0, 9.6, 13.2, 16.4, 24.4, 32.0, 48.0, 64.0, 96.0 or 128.0 Kbit/s. An AMR-WB Interoperable mode is also provided which operates at bit rates for the encoded bit stream of 6.6, 8.85, 12.65, 14.25, 15.85, 18.25, 19.85, 23.05 or 23.85 Kbit/s.

CodecPro's EVS fixed-point implementation is based on the 3GPP TS 26.452 [4] specification whereas its floating-point implementation is based on the TS 26.443 [3] specification.

Memory usage

Memory requirements are expressed in Kbytes.

Memory requirements for ARM Cortex A53 (Aarch64)

ROM

	Code	Data	Total
Encoder	796	18	814
Decoder	696	8	704
Common	859	276	1 135

RAM

	Stack	Heap	Scratch	Total
Encoder	12	80	100	192
Decoder	21	142	80	243

Complexity

Complexity varies depending on the encoding mode selected.

CPU utilization on Cortex A53 (Aarch64)

The measurements were performed using the Linux clock_gettime (CLOCK_MONOTONIC, x) system function under the 64-bit Genntoo operating system running on the Raspberry Pi 3 (Cortex A53 @ 1.2 gHz, ARMv8-A, 1gB SDRAM) development board.

EVS Encoder and Decoder complexity on Cortex A53 – 64-bit (MHz)

Test parameters						Average CPU Utilization		
Sampling Rate (kHz)	BW	input type	bit-rate bps	DTX	channel Aware	Encoder	Decoder	Enc+Dec
16	WB-IO	VoIP	6600	yes		89	75	164
16	WB-IO	VoIP	8850	yes		102	73	176
16	WB-IO	VoIP	12650	yes		118	73	191
16	WB-IO	VoIP	14250	yes		125	73	197
16	WB-IO	VoIP	15850	yes		125	73	198
16	WB-IO	VoIP	18250	yes		128	73	200
16	WB-IO	VoIP	19850	yes		130	73	203
16	WB-IO	VoIP	23050	yes		130	72	202
16	WB-IO	VoIP	23850	yes		150	76	226
8	NB	VoIP	5900	yes		118	67	185
8	NB	VoIP	8000	yes		142	64	206
8	NB	VoIP	9600	yes		159	59	218
8	NB	VoIP	13200	yes		155	65	220
8	NB	VoIP	16400	yes		156	56	212
8	NB	VoIP	24400	yes		151	57	208
8	NB	VoIP	13200	NO	yes	180	69	248
8	NB	VoIP	13200	yes	yes	155	65	220
16	WB	VoIP	5900	yes		150	91	241
16	WB	VoIP	8000	yes		158	94	253
16	WB	VoIP	9600	yes		179	69	248
16	WB	VoIP	13200	yes		180	70	250
16	WB	VoIP	16400	yes		174	61	235
16	WB	VoIP	24400	yes		178	62	240
16	WB	VoIP	13200	NO	yes	236	76	312
16	WB	VoIP	13200	yes	yes	216	69	285
16	WB	Streaming	16400	yes		167	70	236
16	WB	Streaming	24400	yes		172	81	253
16	WB	Streaming	48000	yes		121	74	195
16	WB	Streaming	64000	yes		152	51	203
16	WB	Streaming	128000	yes		122	76	198

32	SWB	Streaming	16400	yes	199	110	309
32	SWB	Streaming	24400	yes	207	110	317
32	SWB	Streaming	48000	yes	166	112	279
32	SWB	Streaming	64000	yes	178	81	259
32	SWB	Streaming	128000	yes	157	105	262
48	FB	Streaming	16400	yes	215	141	356
48	FB	Streaming	24400	yes	225	142	367
48	FB	Streaming	48000	yes	185	181	366
48	FB	Streaming	64000	yes	190	94	285
48	FB	Streaming	128000	yes	184	141	325
16	WB	Streaming	16400	NO	168	72	241
16	WB	Streaming	24400	NO	174	73	247
16	WB	Streaming	48000	NO	123	75	198
16	WB	Streaming	64000	NO	160	51	211
16	WB	Streaming	128000	NO	123	77	200
32	SWB	Streaming	16400	NO	202	113	315
32	SWB	Streaming	24400	NO	211	114	325
32	SWB	Streaming	48000	NO	171	114	285
32	SWB	Streaming	64000	NO	187	81	268
32	SWB	Streaming	128000	NO	160	107	267
48	FB	Streaming	16400	NO	219	145	365
48	FB	Streaming	24400	NO	231	146	377
48	FB	Streaming	48000	NO	190	144	334
48	FB	Streaming	64000	NO	200	94	294
48	FB	Streaming	128000	NO	190	143	333

CPU utilization on Intel Core® i7-3770K

The measurements were performed using the rdtsc instruction system function under the 64-bit Windows 10 Pro 22H2 operating system running on the Intel® Core i7-3770 CPU @ 3.50, 16 Gb RAM development PC.

EVS Encoder and Decoder complexity on i7-3770K (MHz)

Test parameters						Average CPU Utilization		
Sampling Rate (kHz)	BW	input type	bit-rate bps	DTX	channel Aware	Encoder	Decoder	Enc+Dec
16	WB-IO	VoIP	6600	yes		15	17	32
16	WB-IO	VoIP	8850	yes		16	18	35
16	WB-IO	VoIP	12650	yes		18	21	39
16	WB-IO	VoIP	14250	yes		19	22	41
16	WB-IO	VoIP	15850	yes		19	22	41
16	WB-IO	VoIP	18250	yes		20	23	42
16	WB-IO	VoIP	19850	yes		20	23	43
16	WB-IO	VoIP	23050	yes		20	23	43
16	WB-IO	VoIP	23850	yes		21	24	45
8	NB	VoIP	5900	yes		21	24	44
8	NB	VoIP	8000	yes		20	24	44
8	NB	VoIP	9600	yes		23	27	50
8	NB	VoIP	13200	yes		23	27	49
8	NB	VoIP	16400	yes		22	28	51
8	NB	VoIP	24400	yes		22	27	49
8	NB	VoIP	13200	NO	yes	25	29	54
8	NB	VoIP	13200	yes	yes	23	27	49
16	WB	VoIP	5900	yes		23	26	49
16	WB	VoIP	8000	yes		21	25	46
16	WB	VoIP	9600	yes		27	31	58
16	WB	VoIP	13200	yes		27	33	60
16	WB	VoIP	16400	yes		27	33	60
16	WB	VoIP	24400	yes		27	35	62
16	WB	VoIP	13200	NO	yes	33	44	77
16	WB	VoIP	13200	yes	yes	31	39	70
16	WB	Streaming	16400	yes		26	32	58
16	WB	Streaming	24400	yes		27	34	61
16	WB	Streaming	48000	yes		23	28	51
16	WB	Streaming	64000	yes		25	30	54
16	WB	Streaming	128000	yes		23	25	48
32	SWB	Streaming	16400	yes		35	42	77
32	SWB	Streaming	24400	yes		36	44	80
32	SWB	Streaming	48000	yes		34	43	78

32	SWB	Streaming	64000	yes	31	37	68
32	SWB	Streaming	128000	yes	30	33	62
48	FB	Streaming	16400	yes	39	47	86
48	FB	Streaming	24400	yes	41	49	90
48	FB	Streaming	48000	yes	38	50	89
48	FB	Streaming	64000	yes	34	40	74
48	FB	Streaming	128000	yes	39	43	82
16	WB	Streaming	16400	NO	26	33	59
16	WB	Streaming	24400	NO	27	35	62
16	WB	Streaming	48000	NO	23	29	52
16	WB	Streaming	64000	NO	26	31	57
16	WB	Streaming	128000	NO	23	25	48
32	SWB	Streaming	16400	NO	35	43	78
32	SWB	Streaming	24400	NO	37	45	83
32	SWB	Streaming	48000	NO	35	45	80
32	SWB	Streaming	64000	NO	33	39	71
32	SWB	Streaming	128000	NO	30	33	63
48	FB	Streaming	16400	NO	40	49	88
48	FB	Streaming	24400	NO	42	51	93
48	FB	Streaming	48000	NO	39	52	92
48	FB	Streaming	64000	NO	35	41	77
48	FB	Streaming	128000	NO	40	40	85

CPU utilization on Intel® Xeon® W-2265 CPU under Windows 11

The measurements were performed using the rdtsc instruction system function under the 64-bit Windows 11 Pro 22H2 operating system running on the Intel® Xeon(R) W-2265 CPU @ 3.50GHz, 128 Gb RAM development PC.

EVS Encoder and Decoder complexity on Xeon W-2265 – Windows (MHz)

Test parameters						Average CPU Utilization		
Sampling Rate (kHz)	BW	input type	bit-rate bps	DTX	channel Aware	Encoder	Decoder	Enc+Dec
16	WB-IO	VoIP	6600	yes		12	9	21
16	WB-IO	VoIP	8850	yes		18	9	26
16	WB-IO	VoIP	12650	yes		21	8	29
16	WB-IO	VoIP	14250	yes		23	8	31
16	WB-IO	VoIP	15850	yes		23	8	32
16	WB-IO	VoIP	18250	yes		24	9	33
16	WB-IO	VoIP	19850	yes		25	9	33
16	WB-IO	VoIP	23050	yes		25	9	33
16	WB-IO	VoIP	23850	yes		27	9	36
8	NB	VoIP	5900	yes		29	12	41
8	NB	VoIP	8000	yes		19	9	29
8	NB	VoIP	9600	yes		26	8	34
8	NB	VoIP	13200	yes		26	9	35
8	NB	VoIP	16400	yes		20	7	27
8	NB	VoIP	24400	yes		20	7	27
8	NB	VoIP	13200	NO	yes	27	10	37
8	NB	VoIP	13200	yes	yes	26	9	35
16	WB	VoIP	5900	yes		33	14	47
16	WB	VoIP	8000	yes		21	11	32
16	WB	VoIP	9600	yes		30	10	40
16	WB	VoIP	13200	yes		31	10	41
16	WB	VoIP	16400	yes		25	9	34
16	WB	VoIP	24400	yes		25	9	34
16	WB	VoIP	13200	NO	yes	32	10	42
16	WB	VoIP	13200	yes	yes	30	10	40
16	WB	Streaming	16400	yes		24	9	33
16	WB	Streaming	24400	yes		25	9	33
16	WB	Streaming	48000	yes		21	8	28
16	WB	Streaming	64000	yes		23	9	31
16	WB	Streaming	128000	yes		20	9	29
32	SWB	Streaming	16400	yes		33	13	47
32	SWB	Streaming	24400	yes		35	14	49
32	SWB	Streaming	48000	yes		32	12	44

32	SWB	Streaming	64000	yes	30	11	41
32	SWB	Streaming	128000	yes	25	12	39
48	FB	Streaming	16400	yes	39	18	57
48	FB	Streaming	24400	yes	40	18	59
48	FB	Streaming	48000	yes	38	15	53
48	FB	Streaming	64000	yes	33	12	46
48	FB	Streaming	128000	yes	38	19	57
16	WB	Streaming	16400	NO	24	8	33
16	WB	Streaming	24400	NO	25	8	34
16	WB	Streaming	48000	NO	21	7	28
16	WB	Streaming	64000	NO	24	8	32
16	WB	Streaming	128000	NO	20	9	29
32	SWB	Streaming	16400	NO	34	13	47
32	SWB	Streaming	24400	NO	36	14	49
32	SWB	Streaming	48000	NO	33	11	45
32	SWB	Streaming	64000	NO	31	11	42
32	SWB	Streaming	128000	NO	27	12	39
48	FB	Streaming	16400	NO	40	18	57
48	FB	Streaming	24400	NO	42	18	60
48	FB	Streaming	48000	NO	40	15	54
48	FB	Streaming	64000	NO	34	12	46
48	FB	Streaming	128000	NO	39	19	58

CPU utilization on Intel® Xeon® W-2265 CPU under Linux

The measurements were performed using the Linux clock_gettime (CLOCK_MONOTONIC, x) system function under the 64-bit CentOS 7.5 operating system running on the Intel® Xeon(R) W-2265 CPU @ 3.50GHz, 128 Gb RAM development PC.

EVS Encoder and Decoder complexity on Xeon W-2265 – Linux (MHz)

Test parameters						Average CPU Utilization		
Sampling Rate (kHz)	BW	input type	bit-rate bps	DTX	channel Aware	Encoder	Decoder	Enc+Dec
16	WB-IO	VoIP	6600	yes		10	7	17
16	WB-IO	VoIP	8850	yes		13	7	20
16	WB-IO	VoIP	12650	yes		14	6	20
16	WB-IO	VoIP	14250	yes		15	6	21
16	WB-IO	VoIP	15850	yes		15	6	21
16	WB-IO	VoIP	18250	yes		16	6	22
16	WB-IO	VoIP	19850	yes		16	6	22
16	WB-IO	VoIP	23050	yes		16	6	22
16	WB-IO	VoIP	23850	yes		17	6	23
8	NB	VoIP	5900	yes		16	9	25
8	NB	VoIP	8000	yes		14	8	22
8	NB	VoIP	9600	yes		17	7	24
8	NB	VoIP	13200	yes		17	8	25
8	NB	VoIP	16400	yes		16	6	21
8	NB	VoIP	24400	yes		15	6	21
8	NB	VoIP	13200	NO	yes	19	8	27
8	NB	VoIP	13200	yes	yes	17	8	25
16	WB	VoIP	5900	yes		19	10	29
16	WB	VoIP	8000	yes		15	9	24
16	WB	VoIP	9600	yes		21	9	30
16	WB	VoIP	13200	yes		21	9	30
16	WB	VoIP	16400	yes		19	7	27
16	WB	VoIP	24400	yes		20	7	26
16	WB	VoIP	13200	NO	yes	26	9	35
16	WB	VoIP	13200	yes	yes	23	9	32
16	WB	Streaming	16400	yes		19	7	26
16	WB	Streaming	24400	yes		20	7	27
16	WB	Streaming	48000	yes		17	7	24
16	WB	Streaming	64000	yes		19	7	26
16	WB	Streaming	128000	yes		16	8	24
32	SWB	Streaming	16400	yes		27	11	28
32	SWB	Streaming	24400	yes		28	12	40
32	SWB	Streaming	48000	yes		27	10	37

32	SWB	Streaming	64000	yes	25	9	34
32	SWB	Streaming	128000	yes	21	11	32
48	FB	Streaming	16400	yes	33	16	48
48	FB	Streaming	24400	yes	34	16	50
48	FB	Streaming	48000	yes	32	13	44
48	FB	Streaming	64000	yes	28	11	39
48	FB	Streaming	128000	yes	30	15	45
16	WB	Streaming	16400	NO	20	7	27
16	WB	Streaming	24400	NO	20	7	27
16	WB	Streaming	48000	NO	17	6	24
16	WB	Streaming	64000	NO	20	7	27
16	WB	Streaming	128000	NO	16	8	24
32	SWB	Streaming	16400	NO	28	11	39
32	SWB	Streaming	24400	NO	29	12	41
32	SWB	Streaming	48000	NO	28	10	37
32	SWB	Streaming	64000	NO	26	9	35
32	SWB	Streaming	128000	NO	22	11	32
48	FB	Streaming	16400	NO	33	16	49
48	FB	Streaming	24400	NO	35	16	52
48	FB	Streaming	48000	NO	33	12	45
48	FB	Streaming	64000	NO	29	11	40
48	FB	Streaming	128000	NO	31	15	46

Data input/output format

The input to the encoder and output from the decoder is in 16-bit linear PCM, mono, at a sampling frequency of 8000, 16000, 32000 or 48000 Hz.

Bitstream frame layout (bit-packing)

The EVS encoder generates AMR-WB IO or EVS encoded frames as per the configuration parameters described in the next sections of this document. The format of the encoder output and the decoder input speech frames is as per clause A.2.6.2 in [6] and can be used to construct RTP packets. Each encoded speech frame is therefore composed of a ToC byte with the H and F bits always set to 0 (see [6] clause A.2.2.1.2) followed by the encoded parameter indices (see [6] clause 7).

Table 1. Speech frame type to size table

Type	ToC E bit	ToC FT	Description	Frame size in octets including the one octet ToC header
6600	1	0	AMR-WB IO mode	18
8850	1	1	AMR-WB IO mode	24
12650	1	2	AMR-WB IO mode	33
14250	1	3	AMR-WB IO mode	37
15850	1	4	AMR-WB IO mode	41
18250	1	5	AMR-WB IO mode	47
19850	1	6	AMR-WB IO mode	51
23050	1	7	AMR-WB IO mode	59
23850	1	8	AMR-WB IO mode	61
SID	1	9	AMR-WB IO mode	6
LOST	1	14	AMR-WB IO mode	1
NO DATA	1	15	AMR-WB IO mode	1
5900	0	0	EVS ACELP (VBR)	8 to 21
7200	0	1	EVS ACELP	19
8000	0	2	EVS ACELP	21
9600	0	3	EVS ACELP	25
13200	0	4	EVS ACELP	34
16400	0	5	EVS ACELP	42
24400	0	6	EVS ACELP	62
32000	0	7	EVS ACELP	81
48000	0	8	EVS ACELP	121
64000	0	9	EVS ACELP	161
96000	0	10	EVS ACELP	241
128000	0	11	EVS ACELP	321
SID	0	12	EVS ACELP	7
LOST	0	14	EVS ACELP	1
NO DATA	0	15	EVS ACELP	1

EVS API data structures

This section contains definitions of the options and stream information structures.

evs_encoderOption

Description

Codec options, which are passed to the encoder through the `evs_encoderOption` data structure. These are input parameters.

Syntax

```
typedef struct {
      Word32 input_Fs;
      Word32 bitstreamformat;
      Word32 total_brate;
      Word32 Opt_DTX_ON;
      Word32 interval_SID;
      Word32 Opt_RF_ON;
      Word32 rf_fec_offset;
      Word32 rf_fec_indicator;
      Word32 max_bwidth;
} evs_encoderOption;
```

Members

`Input_Fs`	Sampling rate of raw pcm audio signals in Hz {8000, 16000, 32000 or 48000}. (default = 16000)
`bitstreamformat`	Internal use only for testing purposes.
`total_brate`	Bitrate selection in kbps of the codec {5900, 6600, 7200, 8000, 8850, 9600, 12650, 13200, 14250, 15850, 16400, 18250, 19850, 23050, 23850, 24400, 32000, 48000, 64000, 96000 or 128000}
`Opt_DTX_ON`	DTX operation {disabled if 0, enabled otherwise}. (default = 0)
`interval_SID`	SID update interval. Active when `Opt_DTX_ON` is enabled. {0 for variable CNG update interval; 3 to 100 for fixed CNG update interval} (default = 8)
`Opt_RF_ON`	Redundant Frame operation or channel aware mode. Channel-aware mode is supported only when `total_brate` is set at ACELP_13k20. {disabled if 0, enabled otherwise}. (default = 0)
`rf_fec_offset`	Redundant Frame - Frame Erasure Concealment - Offset. Active when `Opt_RF_ON` is enabled. The difference in time units (20 ms) between the transmit time of the primary copy of a frame (n) and the transmit time of the redundant copy of that frame which is piggy backed onto a future frame (n + X) is called the FEC offset X. The optimal FEC offset is a value which maximizes the probability of availability of a partial redundant copy when there is a frame loss at the receiver. {0, 2, 3, 5 or 7} (default= 3)

evs_encoderOption

Members (continued)

rf_fec_indicator	Redundant Frame - Frame Erasure Concealment - Indicator. Active when Opt_RF_ON is enabled. This is the Frame Erasure Rate indicator. Set to LOW for FER rates <5% or to HIGH for FER>5%. This parameter controls the threshold used to determine whether a particular frame is critical or not. Such an adjustment of the criticality threshold is used to control the frequency of partial copy transmission. The high setting adjusts the criticality threshold to classify more frames as critical to transmit as compared to the low setting. {0 - LOW; 1 - HIGH} (default = 1 for HIGH)
max_bwidth	Bandwidth limitation. {0 (NB), 1 (WB), 2 (SWB) or 3 (FB)} (default = 2 for SWB)

evs_decoderOption

Description

Codec options, which are passed to the decoder through the evs_decoderOption data structure. These are input parameters.

Syntax

```
typedef struct {
      Word32 output_Fs;
      Word32 bitstreamformat;
      Word32 amrwb_rfc4867_flag;
      Word32 Opt_VOIP;
      Word32 jbmSafetyMargin;
} evs_decoderOption;
```

Members

Output_Fs	Sampling rate of raw pcm audio signals in Hz {8000, 16000, 32000 or 48000}. (default = 16000)
Bitstreamformat	Bitstream format. {0 (G192), 1 (MIME) or 3 (VOIP_RTPDUMP)} (default = 1 MIME)
Amrwb_rfc4867_flag	Use MIME rfc4867 ToC byte. Active when Opt_VOIP is disabled. {0 for MIME EVS, MIME rfc4867 otherwise} (default = 0)
Opt_VOIP	VoIP and jitter buffer processing. {disabled if 0; enabled otherwise} (default = 0)
jbmSafetyMargin	Jitter Buffer Management Safety Margin during RTP packet payload decoding. Active when Opt_VOIP is enabled. Allowed delay reserve in addition to network jitter to reduce late-loss. (default = 60 ms)

evs_streamInfo

Description

Stream information reported by `D_IF_evs_GetFrameInfo()`

Syntax

```
typedef struct {
      Word32 dec_delay;
      Word32 bitrate;
      Word32 frame_size;
      Word32 jitterBufferEmpty;
      Word32 optimum_offset;
      Word32 FEC_hi;
} evs_streamInfo;
```

Members

`dec_delay`	Decoder compensation delay in PCM samples.
`bitrate`	Bitrate of speech frame.
`frame_size`	The number of octets in speech frame including the ToC.
`jitter_buffer_empty`	0 if jitter buffer is not empty; jitter buffer is empty otherwise.
`optimum_offset`	Optimum redundant frame offset (2, 3, 5 or 7)
`FEC_hi`	Frame Erasure Rate Indicator (0 for FER > 5%, 1 for FER > 5%). See EVS VoIP and Jitter Buffer Details.

EVS API functions

D_IF_evs_queryBlockSize

Description

Outputs the sizes of state and scratch memory blocks required by the decoder.

Syntax

```
#include "typedef.h"
#include "evs_if.h"

void D_IF_evs_queryBlockSize(
       void *iSizeState,
       void *iSizeScratch
);
```

Arguments

iSizeState	(Output) Size of state memory block required by the decoder in bytes.
iSizeScratch	(Output) Size of scratch memory block required by the decoder in bytes.

Return value

0

D_IF_evs_init

Description

Initializes the resources needed for a new instance of the decoder. To “reset” the decoder during normal operation, call this function again.

Syntax

```
#include "typedef.h"
#include "evs_if.h"

Word32 D_IF_evs_init(
       void *hState,
       void *hScratch,
       evs_decoderOption *stDecoderOption,
);
```

Arguments

hState	(Input) Pre-allocated state memory by the host of size defined by a call to D_IF_evs_queryBlockSize(). This memory is unique to each instance of the decoder.
hScratch	(Input) Pre-allocated scratch memory by the host of size defined by a call to D_IF_evs_queryBlockSize().This memory block does not need to be kept unchanged between the API calls.
stDecoderOption	(Input) Pointer to decoder option structure.

Return value Error code if failed, 0 otherwise.

D_IF_evs_decode

Description Decodes one frame of encoded bitstream data.

Syntax

```
#include "typedef.h"
#include "evs_if.h"

Word32 D_IF_evs_decode(
      void   *hState,
      evs_decoderOption *stDecoderOption,
      UWord8 *pBitstream,
      Word16 *pSynthSpeech,
      Word32 *iOutputSize,
      Word16 iBFI
);
```

Arguments

`hState`	(Input) Handle to decoder state memory.
`stDecoderOption`	(Input) Pointer to decoder option structure.
`pBitstream`	(Input) Buffer containing one packet of encoded bitstream data.
`pSynthSpeech`	(Output) Buffer containing one packet of speech samples.
`iOutputSize`	(Output) Number of samples written to `pSynthSpeech`.
`iBFI`	(Input) Bad frame indicator is used to inform the decoder of a frame lost. The input bitstream is ignored and concealment based on the previous frames decoded is performed. Another method to trigger the decoder frame loss concealment procedure is to inject a SPEECH LOSS frame for decoding.

Return value Error code if failed, 0 otherwise.

D_IF_evs_getFrameInfo

Description

Gets information about the frame based on input header.

Syntax

```
#include "typedef.h"
#include "evs_if.h"

Word32 D_IF_evs_getFrameInfo(
      Void *hState,
      evs_decoderOption *stDecoderOption,
      UWord8 *pBitstreamheader,
      evs_streamInfo *stStreamInfo
);
```

Arguments

hState	(Input) Handle to decoder state memory.
stDecoderOption	(Input) Pointer to decoder option structure.
pBitstreamheader	(Input) Pointer to the bitstream header to analyze.
stStreamInfo	(Output) Pointer to the stream info structure.

Return value

Error code if failed, 0 otherwise.

D_IF_evs_close

Description

Close decoder. Free up resources.

Syntax

```
#include "typedef.h"
#include "evs_if.h"

Word32 D_IF_evs_close(
      Void *hState
);
```

Arguments

hState	(Input) Handle to decoder state memory.

Return value

Error code if failed, 0 otherwise.

D_IF_evsPayload_unpackFrame

Description

Sets the output parameter `framePtr` to point at the next EVS frame data of the input payload pointed by `payload` and outputs the frame information using the different output parameters which can be used to create a TOC byte required by the EVS decoder.
`D_IF_evsPayload_unpackFrames` sets `framePtr` to the next EVS frame in time which is specified by the `frameIndex` input parameter. The host is expected to manage `frameIndex` according to the output parameter `frameFollowing`. The input payload is expected to conform to the EVS RTP Payload Format as per clause A.2 of TS 26.445 [6].

Syntax

```
#include "typedef.h"
#include "evs_if.h"

Word32 D_IF_evs_decode(
      Word32   hf_only,
      Const Word8 *payload,
      UWord16  payloadSizeBytes,
      UWord16  frameIndex,
      Word32  *frameFollowing,
      UWord8  *toc_byte,
      UWord8 **framePtr,
      UWord16 *frameSizeBits
);
```

Arguments

`hf_only`	(Input) When set to 1, the routine will not attempt to unpack in Compact format.
`payload`	(Input) Pointer to start of payload.
`payloadSizeBytes`	(Input) Payload size in bytes.
`frameIndex`	(Input) The frame index within the payload that needs to be processed.
`frameFollowing`	(Output) Set to one if the current frame is not the last frame in the payload, set to 0 otherwise.
`toc_byte`	(Output) Set to the value of the Table Of Content byte according to the frame description.
`framePtr`	(Output) Points to the start of the current EVS data frame in the payload.
`frameSizeBits`	(Output) Set to the size of the current EVS data frame in the payload.

Return value

Error code if failed, 0 otherwise.

E_IF_evs_queryBlockSize

Description

Outputs the sizes of state and scratch memory blocks required by the encoder.

Syntax

```
#include "typedef.h"
#include "evs_if.h"

void E_IF_evs_queryBlockSize(
       void *iSizeState,
       void *iSizeScratch
);
```

Arguments

iSizeState	(Output) Size of state memory block required by the encoder in bytes.
iSizeScratch	(Output) Size of scratch memory block required by the encoder in bytes.

Return value

0

E_IF_evs_init

Description

Initializes the resources needed for a new instance of the encoder. To “reset” the encoder during normal operation, call this function again.

Syntax

```
#include "typedef.h"
#include "evs_if.h"

Word32 E_IF_evs_init(
       void *hState,
       void *hScratch,
       evs_encoderOption *stEncoderOption,
);
```

Arguments

hState	(Input) Pre-allocated state memory by the host of size defined by a call to `E_IF_evs_queryBlockSize()`. This memory is unique to each instance of the encoder.
hScratch	(Input) Pre-allocated scratch memory by the host of size defined by a call to `E_IF_evs_queryBlockSize()`. This memory block does not need to be kept unchanged between the API calls.
stEncoderOption	(Input) Pointer to encoder option structure.

Return value

Error code if failed, 0 otherwise.

E_IF_evs_encode

Description

Encodes one frame of 16-bit linear PCM speech data.

Syntax

```
#include "typedef.h"
#include "evs_if.h "

Word32 E_IF_evs_encode(
      void   *hState,
      evs_encoderOption*stEncoderOption,
      Word16 *pSpeech,
      UWord8 *pBitstream,
      Word32 *iOutputSize
);
```

Arguments

hState	(Input) Handle to encoder state memory.
stEncoderOption	(Input) Pointer to encoder option structure.
pSpeech	(Input) Buffer containing one frame of speech samples.
pBitstream	(Output) Buffer containing one frame of encoded bitstream data.
iOutputSize	(Output) Size of the output bitstream in bytes.

Return value

Error code if failed, 0 otherwise.

E_IF_evs_getFrameInfo

Description

Returns the size of the expected encoded output frame and the encoder delay information as per the encoder option settings.

Syntax

```
#include "typedef.h"
#include "evs_if.h"

Word32 E_IF_evs_getFrameInfo(
      evs_encoderOption *stEncoderOption,
      Word32 *iEncoderDelay,
      Word32 *iFrameSize
);
```

Arguments

stEncoderOption	(Input) Pointer to encoder option structure.
iEncoderDelay	(Output) Algorithmic delay of the encoder in number of pcm samples. Useful to align bitstream with the input signal. Normally used to testing purposes.
iFrameSize	(Output) Number of encoded octets to encode one frame as per stEncoderOption. (see **Table 1**)

Return value

Error code if failed, 0 otherwise.

E_IF_evs_close

Description

Close encoder. Free up resources.

Syntax

```
#include "typedef.h"
#include "evs_if.h"

Word32 E_IF_evs_close(
      Void *hState
);
```

Arguments

hState	(Input) Handle to encoder state memory.

Return value

Error code if failed, 0 otherwise.

E_IF_evsPayload_packFrames

Description

Transforms the data of the input bitstream pointed by `bitstream` to one RTP payload as per clause A.2 of TS 26.445 [6]. The input bitstream is expected to be a series of consecutive speech frames (output of `E_IF_evs_encode`) that conform to the Bitstream frame layout section in this document. The first EVS frame of the input bitstream is the first frame in time. Please refer to Encoder RTP Packing Considerations.

Syntax

```
#include "typedef.h"
#include "evs_if.h "

Word32 E_IF_evs_encode(
      UWord8 *bitstream,
      UWord32 cmr_fmtp,
      UWord32 cmr_value,
      Word32  hf_only,
      Word32  inputSize,
      Word32 *outputSize
);
```

Arguments

`bitstream`	(In/Out) Pointer to the bitstream buffer to transform the encoder output to RTP payload ready format.
`cmr_fmtp`	(Input) Set to -1 for CMR disabled except for AMR-WB IO bit-rates and NO_REQ; set to 0 for CMR enabled but not mandatory; set to 1 for CMR mandatory on every payload.
`cmr_value`	(Input) CMR value as per TS 26.445 table A.3; when `cmr_fmtp` is set to 0, set the most significant bit of `cmr_value` to force a CMR on the next call to `E_IF_evsPayload_packFrames`().
`hf_only`	(Input) Header-Full format only when `hf_only` is set to 1; Compact Format allowed otherwise.
`inputSize`	(Input) Size of the input buffer in bytes.
`outputSize`	(Output) Pointer to size of the output payload in bytes.

Return value

Error code if failed, 0 otherwise.

Error codes

This table lists the encoder error codes.

Encoder Error Codes	Value	Description
EVS_ENC_ERR_NONE	0	No error.
EVS_ENC_ERR_GENERIC	-1	General error.
EVS_ENC_ERR_SID_INTERVAL_OUT_OF_RANGE	-2	`Interval_SID` option out of range.
EVS_ENC_ERR_INVALID_BITRATE	-3	Invalid `total_brate` option.
EVS_ENC_ERR_INVALID_SAMPLINGRATE	-4	Invalid `input_Fs` option.

This table lists the decoder error codes.

Decoder Error Codes	Value	Description
EVS_DEC_ERR_NONE	0	No error.
EVS_DEC_ERR_GENERIC	-1	General error.
EVS_DEC_ERR_FAU	-2	Error in Feeding Access Unit to VoIP jitter buffer.
EVS_DEC_ERR_GS	-3	Error Getting Samples from VoIP jitter buffer.
EVS_DEC_ERR_INVALID_SAMPLINGRATE	-4	Invalid `output_Fs` option.
EVS_DEC_ERR_BAD_HEADER	-5	Invalid frame header in input buffer.

EVS Speech-Audio Codec Details

Introduction

The main intent of this section is to provide more detailed information on the selectable codec options outlined in `evs_encoderOption` and `evs_decoderOption` and the EVS codec's internal state indicators outlined in `evs_streamInfo`. Portions of the text in this section are taken from the 3GPP TS 26.441 and TS 26.445 documentation.

Encoder Control Parameters

The EVS codec includes a number of control parameters that can be changed dynamically during regular operation of the codec, without interrupting the audio stream from the encoder to the decoder. These parameters only affect the encoder since any impact they have on the bitstream is signaled in-band such that a decoder can decode any EVS stream without any out-of-band signaling.

The encoder control parameters are listed below.

Encoded Bitrate Selection [evs_encoderOption.total_brate]

Bitrate selection in Kbps of the encoder. EVS supports the bitrates mentioned in the overview section. (can be changed dynamically).

The coder uses the total bit rate parameter to select the encoding technology as described in the following table.

Value	Description
6600	ACELP core layer at 6.6 kbps (used only in AMR-WB IO mode)
8850	ACELP core layer at 8.85 kbps (used only in AMR-WB IO mode)
12650	ACELP core layer at 12.65 kbps (used only in AMR-WB IO mode)
14250	ACELP core layer at 14.25 kbps (used only in AMR-WB IO mode)
15850	ACELP core layer at 15.85 kbps (used only in AMR-WB IO mode)
18250	ACELP core layer at 18.25 kbps (used only in AMR-WB IO mode)
19850	ACELP core layer at 19.85 kbps (used only in AMR-WB IO mode)
23050	ACELP core layer at 23.05 kbps (used only in AMR-WB IO mode)
23850	ACELP core layer at 23.85 kbps (used only in AMR-WB IO mode)
5900	ACELP core layer at average bitrate of 5.90 kbps (used only in DTX_ON mode)
7200	ACELP core layer at 7.2 kbps
8000	ACELP core layer at 8.0 kbps
9600	ACELP core layer at 9.60 kbps
13200	ACELP core layer at 13.20 kbps
16400	ACELP core layer at 16.40 kbps
24400	ACELP core layer at 24.40 kbps
32000	ACELP or HQ core layer at 32 kbps
48000	ACELP or HQ core layer at 48 kbps
64000	ACELP or HQ core layer at 64 kbps
96000	HQ core at 96 kbps
128000	HQ core at 128 kbps

DTX Operation [evs_encoderOption.Opt_DTX_ON]

DTX operation {disabled if 0, enabled otherwise}. (default = 0).

When the codec is operated in DTX ON mode the codec selects the discontinuous transmission (DTX) mode for frames that are determined to consist of background noise. For these frames a low-rate parametric representation of the signal is transmitted no more frequently than every 8 frames (160ms) when `evs_encoderOption.interval_SID` is set to 0. This results in a lower average bitrate.

SID Update Interval [evs_encoderOption.interval_SID]

SID update interval. Active when Opt_DTX_ON is enabled. {0 for variable CNG update interval; 3 to 100 for fixed CNG update interval} (default = 8).

The CN parameters are transmitted at a fixed or adaptive rate during inactive signal periods using the `evs_encoderOption.interval_SID` parameter. The fixed rate is limited to between 3 and 100

frames. The adaptive rate, in general, is dependent on the background noise characteristics such as the current signal-to-noise ratio (SNR) and is limited to be between 8 and 50.

Redundant Frame Operation [evs_encoderOption.Opt_RF_ON]

Redundant Frame operation or channel aware mode. Channel-aware mode is supported only when `evs_encoderOption.total_brate` is set at 13200 {disabled if 0, enabled otherwise}. (default = 0).

EVS offers partial redundancy-based error robust channel aware mode at 13.2 kbps for both wideband and super-wideband audio bandwidths. Please refer to section VoIP and Jitter Buffer Details below.

Redundant Frame FEC Offset [evs_encoderOption.rf_fec_offset]

Redundant Frame - Frame Erasure Concealment - Offset. Active when `evs_encoderOption.Opt_RF_ON` is enabled. The difference in time units (20 ms) between the transmit time of the primary copy of a frame (n) and the transmit time of the redundant copy of that frame which is piggy backed onto a future frame (n + X) is called the FEC offset X. The optimal FEC offset is a value which maximizes the probability of availability of a partial redundant copy when there is a frame loss at the receiver. {0, 2, 3, 5 or 7} (default= 3) (can be changed dynamically).

Redundant Frame FEC Indicator [evs_encoderOption.rf_fec_indicator]

Redundant Frame - Frame Erasure Concealment - Indicator. Active when `evs_encoderOption.OptRF_ON` is enabled. This is the Frame Erasure Rate indicator. Set to LOW for FER rates <5% or to HIGH for FER>5%. This parameter controls the threshold used to determine whether a particular frame is critical or not. Such an adjustment of the criticality threshold is used to control the frequency of partial copy transmission. The high setting adjusts the criticality threshold to classify more frames as critical to transmit as compared to the low setting. {0 - LOW; 1 - HIGH} (default = 1 for HIGH) (can be changed dynamically).

Bandwidth Limitation [evs_encoderOption.max_bwidth]

Bandwidth limitation. {0 (NB), 1 (WB), 2 (SWB) or 3 (FB)} (default = 2 for SWB) (can be changed dynamically.).

For efficiency purposes the `evs_encoderOption.max_bwidth` is limited internally to NB if `evs_encoderOption.sampling_rate` is set to 8000, to WB if is set to 16000 and to SWB and if set to 32000.

Encoder RTP Packing Considerations

It is important to note that the NO DATA frames generated by the EVS encoder when `evs_encoderOption.Opt_DTX_ON` != 0 are not packed in the payload and are not sent to the remote as per clause A.2 of TS 26.445 [6]. While it is straight forward to deal with DTX when the input of the payload packer is one frame, special care is needed when using DTX with multiple frames per payload.

The payload packer function `E_IF_evsPayload_packFrames` will ignore a NO DATA frame and all frames following, except for the first SID (Silence Indication Descriptor) frame encountered after a NO DATA frame. This is done to adhere to the `evs_encoderOption.interval_SID` setting. Ignoring subsequent frames prevents DATA frames from being sent too early to the remote decoder, which could lead to artifacts (distortions in the audio). However, the ignored DATA frames result is lost audio segments that may be experienced following a silence period.

The recommendation is for the calling function to monitor the output from `E_IF_evs_encode` for NO DATA frames.

Scenario 1: If a NO DATA frame is followed by one or more DATA and/or SID frames

- Do not pack (ignore) the NO DATA frame.
- Pack the following DATA and/or SID frames (call `E_IF_evsPayload_packFrames)` to a payload and then pack the payload to an RTP packet with a timestamp that is 20 ms later than the current timestamp.

Scenario 2: If a SID frame is followed by two NO DATA frames and then one or more DATA and/or SID frame:

- Pack the SID frame with the current timestamp.
- Ignore the two NO DATA frames.
- Pack the following DATA and/or SID frames to a payload and then pack the payload to an RTP packet with a timestamp that is 60 ms later than the current timestamp (20 ms for the SID and an additional 40 ms for the two DATA frame).

These are just two of many possible scenarios. Following this structure can ensure proper handling of frames.

Decoder Control Parameters

The decoder can be operated in RTP packet-based mode for VoIP systems or in EVS frame-based mode for file or frame-oriented systems.

The decoder control parameters are listed below.

Bitstream Format [evs_decoderOption. bitstreamformat]

The `evs_decoderOption.bitstreamformat` option is useful to decode RTPDUMP files. Set `evs_decoderOption.Opt_VOIP` to 1 and `evs_decoderOption.bitstreamformat` to 3 (VOIP_RTPDUMP) to decode .rtpdump files. Otherwise set `evs_decoderOption.bitstreamformat` to 1 (MIME). Value 0 (G192) is for internal testing purposes only.

Refer to the sample application source file `voip_client.c` for more details.

AMR-WB RFC4867Selection [evs_decoderOption.amrwb_rfc4867_flag]

Use MIME rfc4867 ToC byte. Active when `evs_decoderOption.Opt_VOIP` is disabled. {0 for MIME EVS, MIME rfc4867 otherwise} (default = 0).

When flag is set for MIME rfc4867 the EVS decoder can operate in a mode which is fully interoperable with the AMR-WB codec bitstream. Please refer to [4] Appendix A. When set for MIME EVS, the decoder expects the bitstream to be packed as per the bit-packing section above.

VoIP and Jitter Buffer Processing [evs_decoderOption.Opt_VOIP]

VoIP and jitter buffer processing. {disabled if 0; enabled otherwise} (default = 0).

Please refer to section VoIP and Jitter Buffer Details below.

JBM Safety Margin [evs_decoderOption.jbmSafetyMargin]

Jitter Buffer Management Safety Margin during RTP packet payload decoding. Active when `evs_decoderOption.Opt_VOIP` is enabled. Allowed delay reserve in addition to network jitter to reduce late-loss. (default = 60 ms).

Decoder RTP Packet Format

When set for VoIP operations, the `D_IF_evs_decode` API function is used to feed RTP payloads to the jitter buffer or to fetch the next frame for decoding from the jitter buffer and generate the next 20 ms of PCM synthesized signal. It requires a 16-octet header that carries the information related to the current RTP packet. It expects every input packet to respect the following formatting:

Octet Number	Description
0	`System Time`, bits 31 to 24 (MSB). The local system time in milliseconds. The decoder will output 20 ms of PCM data every time `D_IF_evs_decode` is called and the System Time is higher than the RTP Packet Arrival Time (octets 9 to 6). Otherwise, `D_IF_evs_decode` will inject the packet in the jitter buffer queue and will not produce PCM samples.
1	`System Time`, bits 23 to 16.
2	`System Time`, bits 15 to 8.
3	`System Time`, bits 7 to 0 (LSB).
4	`RTP Packet Payload Size`, bits 15 to 8 (MSB).
5	`RTP Packet Payload Size`, bits 7 to 0 (LSB)
6	`RTP Packet Arrival Time`, bits 31 to 24 (MSB). The local system time in milliseconds at which the current packet has arrived at the local equipment. The decoder will output one PCM frame every time `D_IF_evs_decode` is called and the System Time (octets 3 to 0) is higher than the RTP Packet Arrival Time. Otherwise, `D_IF_evs_decode` will inject the packet in the jitter buffer queue and will not produce PCM samples.
7	`RTP Packet Arrival Time`, bits 23 to 16.
8	`RTP Packet Arrival Time`, bits 15 to 8.
9	`RTP Packet Arrival Time`, bits 7 to 0 (LSB).
10	`RTP Sequence Number`, bits 15 to 8 (MSB).
11	`RTP Sequence Number`, bits 7 to 0 (LSB)
12	`RTP Time Stamp`, bits 31 to 24 (MSB).
13	`RTP Time Stamp`, bits 23 to 16.
14	`RTP Time Stamp`, bits 15 to 8.
15	`RTP Time Stamp`, bits 7 to 0 (LSB).
16	`RTP Payload`, first octet. The first AMR-WB or EVS header is located here. Please refer to [6] clause A.2.
…	
16 -1 + n	`RTP Payload`, n^{th} octet.
…	
16 -1 + RTP Packet Payload Size	`RTP Payload`, last octet.

If the current packet's `System Time` is larger or equal to the `RTP Packet Arrival Time`, the decoder will feed the current packet payload to the jitter buffer and will not generate any PCM output. Otherwise, it will ignore the payload in the current packet, fetch the next frame to decompress from the jitter buffer and generate the next 20 ms of synthesized data.

Decoder VoIP and Jitter Buffer Details

Jitter Buffers are required in packet-based communications to smooth the inter-arrival jitter of incoming media packets for uninterrupted playout.

The incorporated decoder jitter buffer solution is used in conjunction with the EVS decoder. It is optimized for the Multimedia Telephony Service for IMS (MTSI) and fulfils the requirements for delay and jitter-induced concealment operations set in [6].

In a VoIP system, packets arrive at the decoder with random jitters in their arrival time. Packets may also arrive out of order at the decoder. Since the decoder expects to be fed a speech packet every 20 ms to output speech samples in periodic blocks, a de-jitter buffer is required to absorb the jitter in the packet arrival time. The larger the size of the de-jitter buffer, the better its ability to absorb the jitter in the arrival time and consequently fewer late arriving packets are discarded. Voice communications is also a delay critical system and therefore it becomes essential to keep the end-to-end delay as low as possible so that a two-way conversation can be sustained.

The adaptive Jitter Buffer Management (JBM) solution reflects the above-mentioned trade-offs. While attempting to minimize packet losses, the JBM algorithm in the receiver also keeps track of the delay in packet delivery as a result of the buffering. The JBM solution suitably adjusts the depth of the de-jitter buffer in order to achieve the trade-off between delay and late losses.

The optional channel aware mode (`evs_decoderOption.Opt_RF_ON`) combines the presence of a de-jitter buffer with partial redundancy coding of a current frame which gets piggy backed onto a future frame. At the receiver, the de-jitter buffer is polled to check if a partial redundant copy of the current lost frame is available in any of the future frames. If present, the partial redundant information is used to synthesize the lost frame which offers significant quality improvements under low to high FER conditions.

The channel aware mode encoder may use the `evs_decoderOption.RF_fec_offset` and `evs_decoderOption.RF_fec_indicator` configuration parameters to adapt its operation to track the channel characteristics seen at the remote receiver. These parameters maybe be computed at the remote receiver and communicated to the encoder via a receiver triggered feedback mechanism as described in clause 10.2 of [13]. Table 1.b under sub-clause 10.2.1.9 defines how to map the `evs_streamInfo.optimum_offset` and the `evs_streamInfo.FEC_hi` receiver state indicators in the RTCP-APP signaling adaptation requests.