DResearch H.263+ Codec

Encoder / Decoder Documentation

Version 1.1

DResearch
Digital Media Systems GmbH
Otto-Schmirgal-Straße 3
D-10319 Berlin

Telefon + 49 30 - 515 932 - 0
Fax + 49 30 - 515 932 - 299
e-Mail contact@dresearch.de
WWW http://www.dresearch.de

Geschäftsführer
Managing Director

Dr. Michael Weber

Handelssitz
Amtsgericht Berlin
Charlottenburg
HRB-Nr. 54412

Bankverbindung
Dresdner Bank AG
BLZ 120 800 00
Konto-Nr. 40 472 475 00

Content

Content....2
Working Modes....3
Supported H.263 Annexes....3
Supported H.263+ Features....4
Supported H.263+ Annexes....4
  Additional Codec Features....4
Supported Architectures....5
Function Interface....5
  Encoder Functions....5
  Decoder Functions....16
  Helper Codec Functions....19
Callbacks....22
Changes between version 1.0 and 1.1....24
Demo-Description....25
  Demo Version Restrictions....25
  Encoder Demo Command Line Parameters....25
  Typical Encoder Command Lines....26
  Decoder Demo Command Line Parameters....26
Index....27

Working Modes

The encoder may run in two different modes. The default mode is H.263 (Version 1) compatible. This mode is active after the initialization. The decoder automatically adopt the mode of the stream which is decoded.

The H.263 mode supports the following resolutions:

Resolution	Width	Height
SQCIF	128	96
QCIF	176	144
CIF	352	288
4CIF	704	576
16CIF	1408	1152

Additionally, the encoder may run in H.263 (Version 2) mode, further called H.263+. The mode switch occurs automatically, if an H.263+ annex is activated.
The encoder can run in either constant or variable bit rate mode.

Please note, that the normal license includes H.263 mode only. H.263+ is available on special request.

Supported H.263 Annexes

The following table shows the supported H.263 annexes:

Annex and Description	Encoder	Decoder	Level
A – IDCT accuracy specification	Except MMX version (*)	Except MMX version (*)	
B – Hypothetical Reference Decoder	Not proven	Did not apply	
C – Continuos Presence Multipoint	Frame based multiplexing implemented	Frame based multiplexing implemented	
D – Unrestricted Motion Vector mode	Implemented	Implemented	2
E – Syntax-based Arithmetic Coding mode	Implemented	Implemented	
F – Advanced Prediction mode	Implemented	Implemented	3
G – PB-frames mode	Not implemented	Not implemented	
H – Forward Error Correction for coded video signal	External means (**)	External means (**)	

(*) Full implementation planned for the next major release
(**) Error correction can be easily done by an additional layer. Ask for the additional library.

Please note, that currently there are no plans to support Annex G.

Supported H.263+ Features

Description	Encoder	Decoder
Custom Picture Format up to 2048x1152	Implemented	Implemented
Custom Clock Rate	Implemented	Implemented

Note, that these features are available in H.263+ mode only.

Supported H.263+ Annexes

Currently, not all H.263+ Annexes are supported. The following table shows the currently implemented Annexes.

Annex and Description	Encoder	Decoder	Level
I – Advanced INTRA Coding mode	Implemented	Implemented	1
J – Deblocking Filter	Implemented	Implemented	1
K – Slice Structured mode	Only as GOB replacement	Except RS submode(*)	2
L – Supplemental Enhancement Information	Not implemented	Info will be decoded (***)	1
M – Improved PB-frames mode	Not implemented	Not Implemented	3
N – Reference Picture Selection mode	Implemented	Not implemented (*)	
O – Temporal, SNR and Spatial Scalability mode	SNR & Spatial implemented	Not implemented	
P – Reference picture Resampling	Only Factor-of-4 (****)	Only Factor-of-4 (****)	2
Q – Reduced Resolution Update	Not implemented	Not implemented	
R – Independent Segment Decoding mode	Not implemented	Not implemented	3
S – Alternative INTER VLC mode	Implemented	Implemented	3
T – Modified Quantization mode	Implemented	Implemented	1

(*) Full implementation planned for the next major release
(**) Error correction can be easily done by an additional layer. Ask for the additional library.
(***) For reaching Level 1, „Full-Frame Freeze Only„ is sufficient. This is decoded.
(****) Factor-of-4 mode is sufficient for Level 2

The next major release will support Appendix III Level 2 (current version did support level 2 except the RS submode). Note, that there are no plans to support the Annexes M, Q, O, and R.

Additional Codec Features

Additionally, the DResearch H.263+ codec supports the following features. Note, that these features are available in both H.263 and H.263+ mode.

	Encoder	Decoder
Color space conversion YUV->RGB		MMX-enhanced version
Additional Trace-Mode	Can be enabled at compile time	Can be enabled at compile time
Additional Debug-Information		Available
Advanced Rate Control (TMN9)	Can be enable at runtime	

Supported Architectures

The DResearch H.263+ codec is currently available for the Intel x86 architecture and for Philips TriMedia TM-1X00. The x86 implementation has support for the MMX/SSE/3DNow! extensions. This support is automatically enabled if the encoder runs on a CPU supporting this extensions but can be disabled at runtime.
Additionally, a ANSI-C implementation is available at special request.

Function Interface

The codec is delivered as a set of 3 C-libraries. The library `ent` contains the encoder, `dnt` the decoder and `codec` the generic helper functions used by both encoder and decoder. The following chapter describes the C-function interface of all three libraries.

Encoder Functions

```
ENC_STATUS *Enc_Open(void);
```

Description:

This function opens a new encoder instance. The current TriMedia implementation does not support more than one instance. This may change in the future. The encoder instance is initialized with the following default values:

- H.263 mode
- No Annexes enabled
- No rate control
- Quantization for INTRA and INTER frames: 13
- Allowed quantization range: 1..31

This function must be called first. If the function succeeded, a `ENC_STATUS` handle is returned. This handle has to be used in the further interface calls.

```
int Enc_Init(
    ENC_STATUS *enc,
    int w, int h,
    PUTBYTE_CB PutByteProc,
    void *context,
    char *name
);
```

Parameters:

`enc`	the encoder handle
`w, h`	width and height of the stream
`PutByteProc`	function which will be called for BYTE output, see Chapter Callbacks
`context`	context parameter for the `PutByteProc`
`name`	parameter HAS TO be NULL in the current version

Description:

This function completes the initialization of the encoder and prepares it for the encoder loop. Call this function after annexes are set and configured, and before starting the encoder loop. If successful, this function returns the error code ERR_OK

```
int Enc_SetFramePoolMode(ENC_STATUS *enc, int use_422);
```

Parameters:

enc	the encoder handle
use_422	the encoder handle

Description:

This function switches the frame buffers used by the encoder into either 4:2:2 or 4:2:0 mode. Prior to version 1.1 the encoder uses always 4:2:2 buffers on the TriMedia so it was possible to use a frame buffer directly as a VI unit input. However, this results in 25% of wasted memory per frame store. So, switching to 4:2:0 can safe some memory especially on embedded systems. However, the video front end must than do 4:2:2 to 4:2:0 conversion.
Note, that this mode inflict the buffers management only, the encoder encodes always 4:2:0 as supposed in H.263.
Note further that the encoder by default runs in 4:2:2 buffer mode.

```
void Enc_Stop(ENC_STATUS *enc);
```

Parameters:

enc	the encoder handle

Description:

This Function stops the encoder loop. It may be called asynchronously by signal handlers. See `Enc_Loop()`.

```
void Enc_Term(ENC_STATUS *enc);
```

Parameters:

enc	the encoder handle

Description:

Terminates the encoder instance. This function must be called last. Note, that the handle is invalid after a this call.

```
int Enc_Loop(ENC_STATUS *enc, unsigned max);
```

Parameters:

`enc`	the encoder handle
`max`	number of frames to encode or 0xFFFFFFFF for infinite encoding

Description:

This function implements the encoder loop. It encodes `max` frames using the current encoder mode. This function returns ERR_OK if it successfully has encoded `max` frames or was stopped by `Enc_Stop()` and ERR_MEM if the frame store is out of memory.

```
void Enc_Error(char *buf, int len, int err);
```

Parameters:

`buf`	a pointer to a character buffer
`len`	the size of the character buffer
`err`	a codec returned error code

Description:

Converts the error code `err` into a human redable string and puts it into the `buf` buffer.

```
BOOL Enc_ChangeStreamRes(ENC_STATUS *enc, STREAM *s, int w, int h);
```

Parameters:

`enc`	the encoder handle
`s`	a handle for the stream that should change the resolution
`w`	the new width
`h`	the new height

Description:

This function changes the resolution of a stream. Users should note, that a resolution change may switch the encoder run into H.263+-Mode if available.

```
BOOL Enc_ChangeStreamClk(ENC_STATUS *enc, STREAM *s, int divisor,
    int conv_code);
```

Parameters:

`enc`	the encoder handle
`s`	a handle for the stream that should change the clock frequency
`divisor`	the new clock rate divisor
`conv_code`	the new conversion code, either `CLK_CONV_FACTOR_1000` or `CLK_CONV_FACTOR_1001`

Description:

This function changes the time reference clock frequency of a stream. This function returns TRUE if the switch was successful. Note, that this function can only be used in H.263+ mode, in H.263 mode it always returns FALSE.

```
void Enc_AdvRateCtrl(ENC_STATUS *enc);
```

Parameters:

`enc`	the encoder handle

Description:

This function enables the usage of the advanced TMN9 bit rate control.

```
void Enc_AltRateCtrl(ENC_STATUS *enc);
```

Parameters:

`enc`	the encoder handle

Description:

This function enables the usage of the alternative TMN9 bit rate control.

```
BOOL Enc_SetQuantInterval(ENC_STATUS *enc, int pq_min, int pq_max);
```

Parameters:

`enc`	the encoder handle
`pq_min`	minimal quantization value
`pq_max`	maximal quantization value

Description:

This function limits the allowed quantization range. Normally, the following range is allowed [1..31]. However, without Annex T, values smaller than 3 may lead to artifacts because of incomplete coefficient range. In H.263 mode the value 4 for the minimal quantization quant is suggested.

```
void Enc_SetDefaultQuant(ENC_STATUS *enc, int i_quant, int p_quant);
```

Parameters:

`enc`	the encoder handle
`i_quant`	default INTRA block quantization value
`p_quant`	default INTER block quantization value

Description:

This function allows the setting of the quantization values for INTRA and INTER blocks. These values are used, if the rate control is disabled, otherwise they are ignored with the advanced (TMN9) rate control .

```
void Enc_SetBitrate(ENC_STATUS *enc, int bitrate);
```

Parameters:

`enc`	the encoder handle
`bitrate`	desired bitrate

Description:

This function sets the desired bitrate. A NULL disables the bitrate control.

```
void Enc_SetIntraInterval(ENC_STATUS *enc, int val);
```

Parameters:

`enc`	the encoder handle
`val`	the INTRA interval

Description:

This function sets the INTRA-frame interval. After every `val` frames, the encoder inserts an INTRA-frame.

```
void Enc_SetMotionVectorSearchRange(ENC_STATUS *enc, int range);
```

Parameters:

`enc`	the encoder handle
`range`	motion vector search range in half pixel units

Description:

This function sets the motion vector search range. Note, that the range may be even more restricted depending on the current setup.

```
void Enc_SetMotionVectorSearchMode(int mode);
```

Parameters:

`mode` select the motion search algorithms. Two algorithms are supported (MVS_STEP and MVS_SNOWFLAKE)

Description:

This function sets the motion vector search algorithm for all encoder instances. The MVS_STEP mode uses much more cpu power, but produces less bits. The MVS_SNOWFLAKE mode uses a fast algorithm, which is less efficient in finding motion vectors, such that more bits are produced.

```
void Enc_SetSyncPointInterval(ENC_STATUS *enc, int val)
```

Parameters:

`enc` the encoder handle
`val` insert a GOB/Slice Synchronization point at every val GOB-border

Description:

This function sets the interval for GOB/SLICE synchronization header insertion. Insertion points can be used for error recovery, but increases the bits needed for encoding a frame. The default value is NULL.

```
STREAM *Enc_GetCurrentStream(ENC_STATUS *enc);
```

Parameters:

`enc` the encoder handle

Description:

This function returns the current STREAM handle.

```
STREAM *Enc_GetCurrentStreamID(ENC_STATUS *enc);
```

Parameters:

`enc` the encoder handle

Description:

This function returns the current STREAM id. It is indented to be used from the GRABBER_INIT() callback.

```
int Enc_SwitchStream(ENC_STATUS *enc, int next_stream);
```

Parameters:

`enc`	the encoder handle
`next_stream`	index of the next stream to be encoded (0-3)

Description:

This function switches the stream that should be encoded next when in Continuous-Multipoint mode (Annex C).

```
int Enc_AllocateStreamBuffer(
    ENC_STATUS *enc, int stream,
    PUTBYTE_CB PutByteProc, void *context);
```

Parameters:

`enc`	the encoder handle
`stream`	index of the stream that requires an unique BIT buffer
`PutByteProc`	function that will be called for BYTE output, see Chapter Callbacks
`context`	context parameter for the `PutByteProc`

Description:

This function allocates a unique BIT buffer for a stream. Normally, all streams use the same BIT buffer, resulting in a „merged“ bitstream. Using this function, the bitstreams can be split.

```
int Enc_DeallocateStreamBuffer(ENC_STATUS *enc, int stream);
```

Parameters:

`enc`	the encoder handle
`stream`	index of the stream that requires a unique BIT buffer

Description:

This function deallocates a unique BIT buffer for a stream.

```
char *Enc_ParseAnnex(ENC_STATUS *enc, char *s);
```

Parameters:

`enc`	the encoder handle
`s`	ASCIIZ-string containing the annexes that should be activated, e.g. „DEF“

Description:

This function should be used to activate annexes. If one annex cannot be set, the function returns an error message. The first character of the error message is the first annex found, which cannot be activated (either not implemented in the DResearch Codec or deactivated). Call this function always before `Enc_ConfigureOptions()`.

```
int Enc_ConfigureOptions(
    ENC_STATUS *enc,
    int w, int h,
    int freq,
    int plus_mode,
    char **err_msg
);
```

Parameters:

`enc`	the encoder handle
`w, h`	width and height of the stream to be encoded
`freq`	possible custom picture clock frequency in Hz
`plus_mode`	wheather H.263+ features should be enabled, one of `AUTO_PLUSMODE, FORCE_PLUSMODE, FORBID_PLUSMODE`
`err_msg`	returns an optional error message

Description:

This function checks whether all selected options will work together. If yes, this function returns ERR_OK and configures the encoder for the selected mode. If not, this functions returns an error code and may set an optional error message.
Always call this function after `Enc_ParseAnnex()`.

```
FRAME_CB Enc_SetFrameCB(ENC_STATUS *enc, FRAME_CB cb, void *context);
```

Parameters:

`enc`	the encoder handle
`cb`	new frame-completion callback
`context`	context parameter, will be passed to the callback

Description:

This function sets a new frame completion callback. The old callback function is returned. The callback function is called by `Enc_Loop()`, whenever a complete frame is encoded.

```
GRABBER_CB Enc_SetGrabberCB(ENC_STATUS *enc, GRABBER_CB cb, void *context);
```

Parameters:

`enc`	the encoder handle
`cb`	new grab-frame callback
`context`	context parameter, will be passed to the callback

Description:

This function sets a new grab-frame callback. The old callback function is returned. The callback function is called, whenever a new frame must be read from the video input.

```
GRAB_INIT_CB Enc_SetGrabInitCB(
    ENC_STATUS *enc,
    GRAB_INIT_CB cb,
    void *context
);
```

Parameters:

`enc`	the encoder handle
`cb`	new grabber init callback
`context`	context parameter, will be passed to the callback

Description:

This function sets a new grabber init callback. The old callback function is returned. The callback function is called, whenever a grabber parameter is changed and before the grabber is called the first time.

```
DWORD Enc_GetTR(STREAM *s);
```

Parameters:

`s`	STREAM handle

Description:

This functions returns the current time in clock-tick of the specified streams clock frequency.

```
void Enc_EnableFastINTER(ENC_STATUS *enc, BOOL flag);
```

Parameters:

`enc`	the encoder handle
`flag`	enables/disables the fast INTER-frame encoder

Description:

This function enables or disables the fast INTER-frame encoder. This mode has several restrictions, but allows the fastest encoding speed. If the fast INTER-frame encoder is activated, rate-control is deactivated and the motion estimations is reduced to the minimum, yielding in a high bit rate.

```
int Enc_SetNotCodedSADThreshold(ENC_STATUS *enc, int threshold)
```

Parameters:

`enc`	the encoder handle
`threshold`	enables/disables the fast INTER-frame encoder

Description:

This function allows to specify a threshold value for the CODED/NOT CODED block decision. By default, the encoder decides it depending on the residual error. By specifying a threshold value, this decision can be made earlier without doing motion estimation for a macroblock thus speeding up the encoding process. However this option may reduce the quality of the encoded frames. Please note that the optimal threshold value may depend on the resolution and on scene details.

```
int Enc_MaxFrameSkip(ENC_STATUS *enc, int max_frame_skip);
```

Parameters:

`enc`	the encoder handle
`max_frame_skip`	maximum number of frames that may be skipped

Description:

If the rate control is limited in ist maximum quantizer or if the bitbudget is too small, the encoder must sometimes skip frames to stay in to allowed bit budget. This function can limit the maximum amount of skipped frames. Please note, that skipping frames is the last possibility for teh encoder to ensure a given bit budget, reducing teh maximum amount of skipped frames will sometimes violate the upper limit for the number of generated bits.

```
FRAME *Enc_GetTestFrame(STREAM *s, FRAME *f, unsigned time_ref);
```

Parameters:

`s`	handle of the current stream
`f`	handle of the current frame
`time_ref`	time reference for this frame

Description:

This function can be used by an application to create a test frame instead to grab one from a grabber front end. This function replaces the version 1.0 test mode of the encoder and should be only called from the grabber callback.

```
BOOL Enc_ForceIntraOnResChange(ENC_STATUS *enc, BOOL flag);
```

Parameters:

`enc`	the encoder handle
`flag`	enables/disables the INTRA-frame-on –resolution-change feature

Description:

This function changes the behaviour of the encoder if the first frame after a resolution change must be encoded. If flag is TRUE, the encoder forces an INTRA frame. If it is FALSE (default), the encoder uses upsample/downsample filter and may encode it as an INTER frame. Note, that this is conform to H.263+, not H.263v1, so one should set it to TRUE if H.263v1 compatibility is requiered.

```
void Enc_GetVersionInfo(version_t *version);
```

Parameters:

`version`	pointer to a `version_t` that will be filled up

Description:

This function returns the current version and build number of the encoder.

Decoder Functions

```
DEC_STATUS *H263P_InitDecoder(FRAME_POOL *p, GetByteProc GetByte, void
*pContext);
```

Parameters:

`p`	a frame pool handle
`GetByte`	the function that should be called for the next input byte
`pContext`	the context parameter for the `GetByte` function

Description:

This function initializes a H.263+ decoder and creates a new decoder instance. This function must be called first in every decoder instance. Note, that different decoder instances may share a frame pool. In that case the user has to ensure that the pool does not run empty.

```
int H263P_GetFrame(DEC_STATUS *s, FRAME **ret_frame, FRAME *last_frame);
```

Parameters:

`s`	decoder instance handle
`ret_frame`	will contain the next decoded frame after a successful return
`last_frame`	the handle of the last decoded frame

Description:

This function contains the main H.263+ parser and decodes the next frame. Because H.263 uses INTER frame prediction, the previous decoded frame must be specified. Note that this information is NOT stored in the decoder status, allowing seeking operations in the data stream. This functions returns a H26X_DEC error code.

```
void H263P_SetCallback(DEC_STATUS *s, LAYER_CB layer_cb);
```

Parameters:

`s`	decoder instance handle
`layer_cb`	layer callback function

Description:

This function sets the layer callback function. The layer callback is called by the decoder, whenever the H.263-layer changes, the substream-ID changes or a different reference frame is specified. The layer callback allows the user to specify a different reference frame depending on the current situation.

```
int H263P_SetErrorRecoveryMode((DEC_STATUS *s, int mode);
```

Parameters:

s	decoder instance handle
mode	new error recovery mode

Description:

This function allows to change the error recovery mode. Error recovery is executed, whenever the decoder detects wrong data in the bitstream. Currently, only the mode ERROR_RECOVERY_COPY_SLICE and 0 are supported. In the copy slice mode, the complete slice (or GOB) from the reference frame is copied to the current frame, the 0 mode does no error recovery. This function returns the old decoder mode.

```
int H263P_DropBits(DEC_STATUS *s);
```

Parameters:

s	decoder instance handle

Description:

This function drops any bit lookahead of the decoder. It must be used, if seek operations are executed on the input data. It returns the number of bits that were dropped.

```
BOOL H263P_TermDecoder(DEC_STATUS *s);
```

Parameters:

s	decoder instance handle

Description:

This function terminates a decoder instance. The frame pool is NOT destroyed.

```
int H263P_GetFrameType(FRAME *f);
```

Parameters:

f	frame handle

Description:

This function returns the frame type of the decoded frame. The following values may be returned:

Value	Meaning
FRAME_TYPE_I	Was codes as an INTRA frame
FRAME_TYPE_P	Was coded as an INTER frame
FRAME_TYPE_PB	Was coded as an PB frame
FRAME_TYPE_EI	Was coded as an enhanced INTRA frame (only in enhaced layer)
FRAME_TYPE_EP	Was coded as an enhanced INTER frame (only in enhaced layer)
FRAME_TYPE_B	Was coded as an B frame (only in enhaced layer)

```
void H263P_GetVersionInfo(version_t *version);
```

Parameters:

`version` pointer to a `version_t` that will be filled up

Description:

This function returns the current version and build number of the decoder.

Helper Codec Functions

```
FRAME_POOL *Hlp_CreatePool(int max_frames)
```

Parameters:

`max_frames` maximum number of frames in the pool

Description:

This function creates a framepool. While every encoder instance creates its own framepool, the user has to create a frame pool for the running decoder depending on the needs. Normally, every decoder instance needs 5 frames for decoding. However, if the application wants to store frames in a frame cache, more frames can be created.

```
void Hlp_TermPool(FRAME_POOL *p);
```

Parameters:

`p` frame pool

Description:

This function terminates a frame pool and releases all allocated resources. Note, that all frames belonging to this pool will be destroyed!

```
PICT *Hlp_Frame2Pict(FRAME *f);
```

Parameters:

`f` FRAME handle

Description:

This function converts a FRAME handle into a PICT structure.

```
FRAME *Hlp_Pict2Frame(PICT *p);
```

Parameters:

`p` PICT structure

Description:

This function converts a PICT structure into a FRAME handle.

```
PICT *Hlp_GetPict(FRAME_POOL *p, int w, int h, int mode);
```

Parameters:

`p`	frame pool
`w`	width
`h`	height
`mode`	one of `FRAME_MODE_*`

Description:

This function allocates a new free PICT structure from the frame pool `p` and sets its reference counter to 1. The parameters `mode` allows to specify the type of the allocated frame. This is useful for grabber front ends, were one would specify `FRAME_MODE_RAW | FRAME_MODE_422` and use it as buffers for the grabbed video data.

Supported modes	
`FRAME_MODE_H261`	Create a frame for the DResearch H.261 codec
`FRAME_MODE_H262`	Create a frame for the DResearch MPEG-II decoder
`FRAME_MODE_H263`	Create a frame for the DResearch H.263+ codec
`FRAME_MODE_MPG4`	Create a frame for the DResearch MPEG-4 codec
`FRAME_MODE_RAW`	Create a RAW frame, suitable for TriMedia VideoIn/VideoOut
`FRAME_MODE_JPG`	Create a frame for the DResearch MotionJPEG codec

Additional memory allocation flags	
`FRAME_MODE_420`	Allocate the buffer in the 4:2:0 format
`FRAME_MODE_422`	Allocate the buffer in the 4:2:2 format (required for TriMedia VideoIn)

```
void Hlp_PutPict(FRAME_POOL *p, PICT *pict);
```

Parameters:

`p`	frame pool
`pict`	PICT structure

Description:

This function decreases the reference counter of a PICT and frees it if the counter reaches 0 to the frame pool `p`. Note that you should NOT release frames to a different frame pool as to which they were allocated. This error condition is not checked.

```
void Hlp_PictAddRef(PICT *pict);
```

Parameters:

`pict`	PICT structure

Description:

This function increases the reference counter of a PICT.

```
void Hlp_InvalidatePict(PICT *pict);
```

Parameters:

`pict` PICT structure

Description:

This function invalidates the PICT in the TriMedia data cache. It is only available on the TriMedia architecture.

```
void Hlp_CopybackPict(PICT *pict);
```

Parameters:

`pict` PICT structure

Description:

This function issues a copyback operation on the PICT data in the TriMedia data cache. It is only available on the TriMedia architecture.

Callbacks

Some of the functionality of the encoder is realized with the help of callback functions. Note that all API functions can be called from any of these callback functions.

```
void (*FRAME_CB)(
      FRAME *curr_frame,
      FRAME *reco_frame,
      int frame_type,
      int bits,
      void *context
);
```

Parameters:

`curr_frame`	the handle of the grabbed frame which was encoded
`reco_frame`	the handle of the reconstructed frame which was encoded
`frame_type`	the chosen frame mode, one of `FRAME_TYPE_*`
`bits`	number of bits spent by the encoder for this frame
`context`	context parameter, free for the user

Desription:

The frame-completion callback is called, whenever the `Enc_Loop()` functions has completed a whole frame.

```
void (*PUTBYTE_CB)(BYTE c, void *context);
```

Parameters:

`c`	byte to be put
`context`	context parameter, free for the user

Desription:

This function is called by the encoder, whenever a BYTE need to be send out. Note that the syntax allows the usage of the standard fputc() function.

```
FRAME *(*GRABBER_CB)(STREAM *s, FRAME *f, int skip_frames, void *context);
```

Parameters:

`s`	handle of the current stream
`f`	handle of the current frame
`skip_frames`	number of frame that should be skipped from the input plus 1
`context`	context parameter, free for the user

Description:

This callback is called by the encoder, whenever a new frame must be read from the video input. The user is responsible for filling the data into the frame. This function returns the frame handle of a filled frame. Note that this can be a different frame for implementing a grabber pipeline. The parameter `skip_frames is` calculated by the rate control and should normally reflect the target skip rate. A grabber front end can ignore this parameter at the cost of creating more bits than selected in the bit rate.

```
void (*GRAB_INIT_CB)(int w, int h, int skip_rate, void *context);
```

Parameters:

`w, h`	width and height of the next frame to be grabbed
`skiprate`	number of frames that should be skipped between two grabbed frames using the current clock rate
`context`	context parameter, free for the user

Description:

This callback is called by the encoder before the first frame is grabbed, or whenever a grabber parameter changes.

```
int   (*GETBYTE_CB)(void *context);
```

Parameters:

`context`	context parameter, free for the user

Desription:

This function is called by the decoder, whenever a BYTE is needed to be read in. Note that the syntax allows the usage of the standard fgetc() function.

```
FRAME *(*LAYER_CB) )(int r_layer, int stream);
```

Parameters:

`r_layer`	requested reference layer
`stream`	requested stream ID

Description:

This callback is called by the decoder whenever the reference layer or the substream ID changes. It must return the associated reference frame.

Changes between version 1.0 and 1.1

The following functions have been added to the encoder Interface:

`Enc_SetFramePoolMode()`	Changes the buffer management of the encoder
`Enc_GetCurrentStreamID`	Returns the stream ID of the current stream
`Enc_MaxFrameSkip()`	Sets the maximal frame skip value for the rate control
`Enc_GetTestFrame()`	Get a test frame created by the encoder instead of grabbing one
`Enc_ForceIntraOnResChange()`	Change the behavior of the encoder after a resolution change

The following functions have been changed:

`Enc_Error()`	Wrote now into a buffer instead of a FILE
`GRABBER_CB()`	Has now additional paramater skip_frames
`Hlp_GetPict()`	Additional parameter mode added

The following functions have been removed:

`Enc_SetTestFrameMode()`	Use `Enc_GetTestFrame()` instead

The following functions have been added to the decoder Interface:

`H263P_GetVersionInfo()`	Returns the version of the decoder

Demo-Description

The DEMO version of the DResearch H.263+ codec for the TriMedia architecture is delivered as 3 little-endian libraries, called ent_demo.a, dnt_demo.a and codec_demo.a. Additionally, the sources for two demo applications are delivered. This demo applications shows the usage of the codec API.

Demo Version Restrictions

The free demo-version has several restrictions:

- All annexes except Annex A und B are deactivated in the encoder
- All H.263+ features are deactivated in the encoder
- Lowest bit rate is 28800 bps
- A copyright message is placed in the center of every encoded and decoded frame

Encoder Demo Command Line Parameters

The encoder demo application has the following command line parameters:

Parameter	Meanning
-a format	format to grab (SQCIF, QCIF, CIF, or width X height)
-f frames	number of frames to grab
-g gob	put every gob a GOB/Slice header, (0 = none)
-n [+|-]	force H.263+ / forbid any H.263+ extensions
-o filename	name of output streamfile
-q quant	starting quality [1-31]
-r bitrate	target bitrate in bits per second (0 switches off)
-s skiprate	target skiprate of stream (framerate=29.97/skiprate), 0 = as fast as possible
-S skiprate	source skiprate of framefile (framerate=29.97/skiprate), 0 = ignore
-v	view supported H.263+ Annex
-A annex(es)	use Annex(es)
-F freq	use custom picture clock frequency freq
-I n	put an Intra-Frame after every n frames
-T	use internally generated test frames
-V range	limit the motion vector range (in halfpixel units)
-W threshold	Threshold value for coded/not coded block decision (disabled by default)
-X	use SnowFlake Motion Vector search instead of StepSearch
-Y	use Advanced TMN9 rate control, else Alternative TMN9
-Z	use the fast INTER-frame encoder
-N	use NTSC else PAL

Please note, that –S0 deactivates the time function, and simulates a compression time of 0. Note further, that all H.263 annexes are disabled in the demo version as well as H.263 custom resolutions.

Typical Encoder Command Lines

```
ent –aCIF –f100 –q4 –otest.263 coast_cif.yuv
```

Encode 100 frames in CIF resolution using quantizer 4, no bit rate control and no time control. Frames will be read from the fyle coast_cif.yuv.

```
ent –aCIF –f100 –S1 –s3 –q4 –otest.263 coast_cif.yuv
```

Encode 100 frames in CIF resolution using quantizer 4, so bit rate control with 29,97/3 framerate. Frames will be read from the fyle coast_cif.yuv.

```
ent –aCIF –r128000 –f100 –S1 –s3 –otest.263 coast_cif.yuv
```

Encode 100 frames in CIF resolution with bit rate 128000 bps with 29,97/3 framerate. Frames will be read from the fyle coast_cif.yuv.

```
ent –aCIF–f100 –Z –q6 –otest.263 coast_cif.yuv
```

Encode 100 frames in CIF resolution with maximum frame rate using the fast INTER-frame encoder. The bitrate is variable.

Decoder Demo Command Line Parameters

The decoder demo application has the following command line parameters:

Parameter	Meanning
-f frames	maximum number of frames to decode
-s num	Start dumping with frame num
-X	Don’t create the seq.yuv output file

The decoder demo decodes a given H.263+ bitstream and appends the output to a YUV concatenated file seq.yuv. The free HEDIT+ Viewer can be used to view the decoded content.

Index

3DNow!....5
Annex
- A – IDCT accuracity specification....3
- B – Hypothetical Reference Decoder....3
- C – Continous Presence Multipoint....3
- D – Unrestricted Motion Vector mode....3
- E – Syntax-based Arithmetic Coding mode....3
- F – Advanced Prediction mode....3
- G – PB-frames mode....3
- H – Forward Error Correction for coded....3
- I – Advanced INTRA Coding mode....4
- J – Deblocking Filter....4
- K – Slice Structured mode....4
- L – Supplemental Enhancement Information....4
- M – Improved PB-frames mode....4
- N – Reference Picture Selection mode....4
- O – Temporal, SNR and Spatial Scalability mode....4
- P – Reference picture Resampling....4
- Q – Reduced Resolution Update....4
- R – Independent Segment Decoding mode....4
- S – Alternative INTER VLC mode....4
- T – Modified Quantization mode....4

Callback
- FRAME_CB()....21
- GETBYTE_CB()....22
- GRAB_INIT_CB()....22
- GRABBER_CB()....21
- LAYER_CB()....22
- PUTBYTE_CB()....21

Colorspace Conversion....4
Enc_ SetSyncPointInterval()....10
Enc_AdvRateCtrl()....8
Enc_AllocateStreamBuffer()....11
Enc_AltRateCtrl()....8
Enc_ChangeStreamClk()....7
Enc_ChangeStreamRes()....7
Enc_ConfigureOptions()....12
Enc_DeallocateStreamBuffer()....11
Enc_EnableFastINTER()....13
Enc_Error()....7
Enc_ForceIntraOnResChange()....15
Enc_GetCurrentStream()....10
Enc_GetCurrentStreamID()....10
Enc_GetTestFrame()....14
Enc_GetTR()....13
Enc_GetVersionInfo()....15
Enc_Init()....5
Enc_Loop()....7
Enc_MaxFrameSkip()....14
Enc_Open()....5
Enc_ParseAnnex()....12
Enc_SetBitrate()....9
Enc_SetDefaultQuant()....9
Enc_SetFrameCB()....12
Enc_SetFramePoolMode()....6
Enc_SetGrabberCB()....13
Enc_SetGrabInitCB()....13
Enc_SetIntraInterval()....9
Enc_SetMotionVectorSearchMode()....10
Enc_SetMotionVectorSearchRange()....9
Enc_SetNotCodedSADThreshold()....14
Enc_SetQuantInterval()....8
Enc_Stop()....6
Enc_SwitchStream()....11
Enc_Term()....6
FRAME_MODE_420....19
FRAME_MODE_422....19
FRAME_MODE_H261....19
FRAME_MODE_H262....19
FRAME_MODE_H263....19
FRAME_MODE_JPG....19
FRAME_MODE_MPG4....19
FRAME_MODE_RAW....19
H.263+ Features
- Custom Clock Rate....4
- Custom Picture Format....4

H263P_DropBits()....17
H263P_GetFrame()....16
H263P_InitDecoder()....16
H263P_SetCallback()....16
H263P_SetErrorRecoveryMode()....17
H263P_TermDecoder()....17
Hlp_CopybackPict()....20
Hlp_CreatePool()....18
Hlp_Frame2Pict()....18
Hlp_GetPict()....19
Hlp_InvalidatePict()....20
Hlp_Pict2Frame()....18
Hlp_PictAddRef()....19
Hlp_PutPict()....19
Hlp_TermPool()....18
Intel x86....5
MMX....5

Philips TriMedia 5
SSE 5
TMN9 4