



ENCIRIS
TECHNOLOGIES

ENCIRIS SOFTWARE DEVELOPER MANUAL

LT-101, LT-102, LT-122, LT-124, LT-200

NOVEMBER 2016

APPLICABLE FROM LIBRARY VERSION 2.87 ON

ENCIRIS LT-XXX CROSS-PLATFORM API DESCRIPTION (WINDOWS, LINUX, MAC OS)

CONTENTS

Enciris LT-xxx Cross-platform API description (Windows, Linux, Mac OS).....	2
Introduction.....	4
Application Flow	5
Global Initialization functions	7
LoadLtDll ().....	8
FreeLtDll ()	9
LT_GetDevicePath()	10
LT_OpenDevice()	11
LT_CloseDevice().....	12
Configuration functions.....	13
LT_SetParam ().....	14
LT_GetParam ()	31
LT_UploadDeviceParameters()	33
LT_GetHighPrecFrInterval()	35
LT_GetVideoDescription()	37
Text overlay	40
LT_SetOsd ().....	41
Control functions.....	43
LT_StartEnc()	44
LT_StopEnc()	45

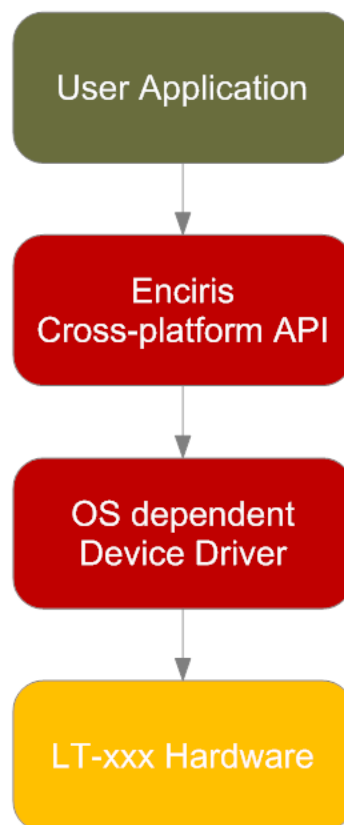
LT_InsertIFrame()	47
LT_InsertSkipFrame ()	48
LT_ForceSkipFrames ().....	49
LT_StartRawCapture()	50
LT_StoptRawCapture().....	52
LT_StartAudio().....	53
LT_StopAudio()	54
Data transfer functions	55
LT_GetEncData()	56
LT_GetEncFrame()	58
LT_GetRawData().....	60
LT_GetRawImage()	63
LT_GetAudio().....	65
ASF/MPEG-TS Multiplexer functions.....	67
LT_InitTsMuxer ().....	68
LT_InitAsfMuxer ()	69
LT_GetMuxedFrame().....	70
LT_MuxFrame().....	73
LT_GetAsfIndexBuffer()	76
LT_GetAsfHeaderObject().....	79
Miscellaneous functions.....	81
LT_SetEdid ()	82
LT_GetEdid ()	85
LT_GetError ()	87
LT_GetVideoInputStatus ()	88
LT_Log ()	92
LT_InitializeDevice ().....	94
LT_GetStillImage ()	95

LT-XXX API ERROR Codes	97
Conclusion	102

INTRODUCTION

This software developer manual is designed for cross platform usability. It will describe *Enciris cross-platform Application Programming Interface (API)* for LT-101, LT-102, LT-122, LT-124, LT-200 framegrabber boards. For Windows, a *Dynamic Link Library (DLL)* is provided which exports the API functions. Linux uses a shared library (.so). Note that an additional *Directshow* library is also provided together with Windows driver for quick application development. This Directshow library is built on top of the Enciris cross-platform API and is extensively described in a dedicated document.

The general architecture of the Enciris cross-platform API is given in the picture below.

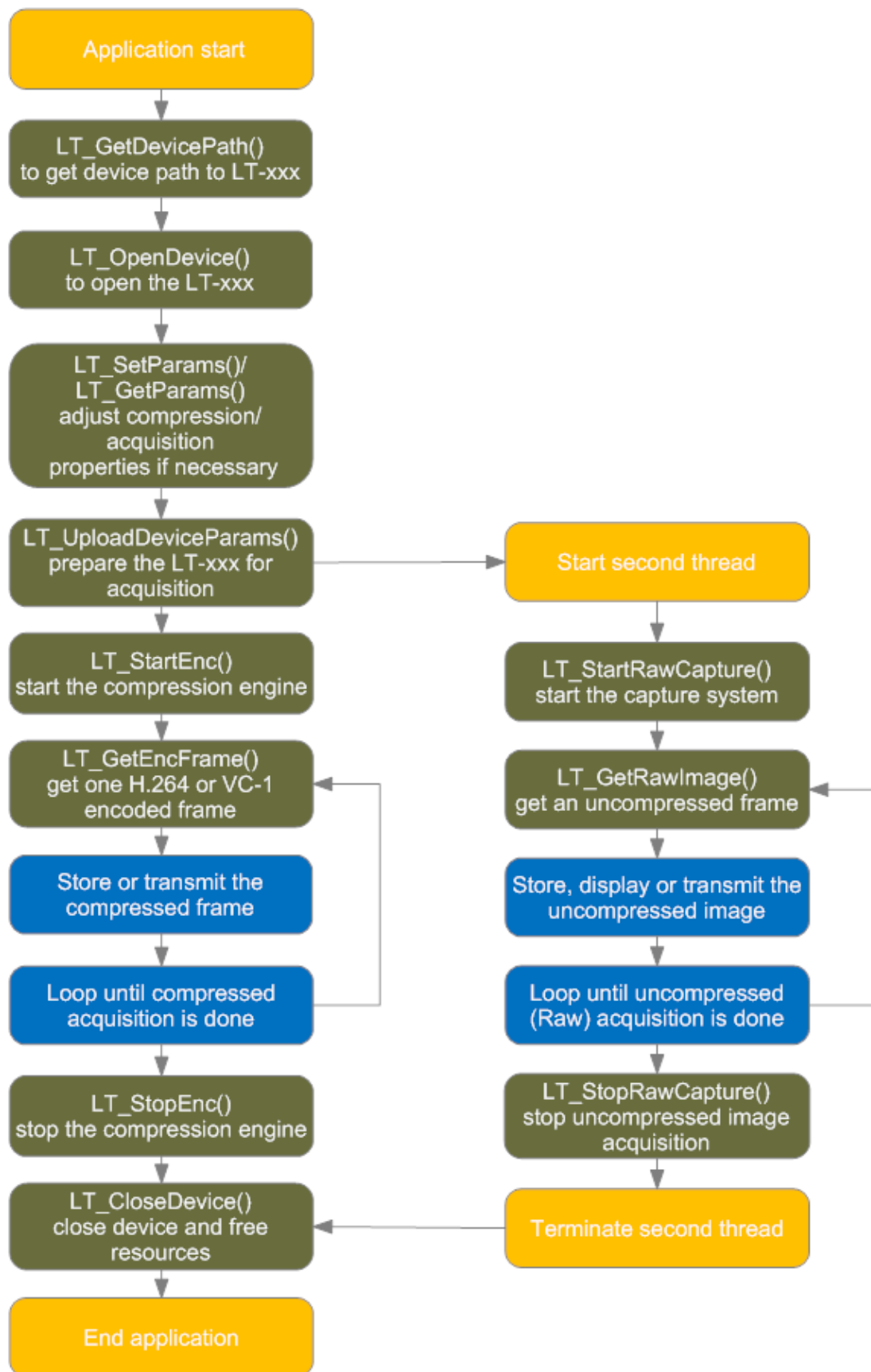


Enciris LT-xxx driver architecture

Important note: The VC-1 driver is a legacy driver and is only provided on demand. From driver version 2.87 on, only H.264 feature will be supported. VC-1 driver can be provided separately but no updates are planned for it.

APPLICATION FLOW

The flow of a typical application is given below:



Global Initialization functions

LOADLTDLL ()

This function opens the "encirislt.dll" Dynamic Link Library (DLL) by calling the underlying Windows "LoadLibrary()" function or the Linux "dlopen()" function. It also gets function pointers to all API functions.

```
HMODULE LoadLtDll ( LPCSTR lpLibFileName);
```

ARGUMENTS

lpLibFileName (in)

Name of the library. Usually "encirislt.dll" under Microsoft Windows or "libencirislt.so" under Linux.

RETURN VALUE

Returns a valid handle to the library on success. A return value of NULL indicates an error.

COMMENTS

This procedure is pretty much similar to other dynamic link library binding schemes.
Note: The DLL must be placed in the same directory than the *.bit Firmware files.

USAGE EXAMPLE

```
HMODULE m_ltdll;  
  
printf("--> LoadLtDll\n");  
m_ltdll = LoadLtDll("encirislt.dll");  
if( !m_ltdll ){printf("Library encirislt.dll not found !\n"); return -1;}
```


FREETDII ()

This function releases the “encirislt.dll” Dynamic Link Library (DLL) by calling the underlying Windows “FreeLibrary()” function or the Linux “dlclose()” function.

```
int FreeLtDII ( HMODULE m_ltDII);
```

ARGUMENTS

m_ltDII (in)
handle to the library opened with “LoadLtDII()”.

RETURN VALUE

Returns 0 on success. A return value of -1 indicates an error.

USAGE EXAMPLE

```
HMODULE m_ltDII;  
  
printf("--> LoadLtDII\n");  
m_ltDII = LoadLtDII("encirislt.dll");  
if( !m_ltDII ){printf("Library encirislt.dll not found !\n"); return -1;}  
...  
...  
FreeLtDII(m_ltDII);
```

LT_GETDEVICEPATH()

This function retrieves a list of the logical pathnames to all the LT-xxx devices installed and operational on host system.

```
int LT_GetDevicePath( char * pDevicePath,  
                     int   DeviceIndex );
```

ARGUMENTS

pDevicePath (out)

A pointer to the device path.

DeviceIndex (in)

This is an index to the card. This number can vary between zero and the total number of LT-xxx devices (31) connected to a PC minus 1.

RETURN VALUE

Returns zero on success. A return value of < 0 indicates an error.

COMMENTS

Every LT-XXX device in the system is given a unique device path. This device path is then used to open the LT-XXX device (see [LT_OpenDevice\(\)](#)).

Hint: The device path string can be used to differentiate between USB and PCI connected LT-XXX devices.

USAGE EXAMPLE

```
char dp[256];  
for( i = 0; i < total_LTXXXs; i++ )  
{  
    if ( LT_GetDevicePath( dp, i ) )  
        return -1;  
    printf( "Device %d: %s\n", i, dp );  
}
```

LT_OPENDEVICE()

This function opens and obtains a handle to a LT-XXX device using the specified device path.

```
Int LT_OpenDevice( char * pDevicePath );
```

ARGUMENTS

pDevicePath (in)

This is a device path obtained from LT_GetDevicePath().

RETURN VALUE

Returns a handle to the opened LT-XXX device on success. A negative return value indicates an error.

COMMENTS

This function opens the driver and establishes communications with the card.

USAGE EXAMPLE

```
char dp[256];
int LtHandle;

if( LT_GetDevicePath( dp, 0 ) )
    return -1;

LtHandle = LT_OpenDevice( dp );

if( LtHandle < 0 )
    return -1;
```

LT_CLOSEDEVICE()

This function closes the LT-XXX device corresponding to the handle passed as parameter.

```
int LT_CloseDevice ( int * LtHandle );
```

ARGUMENTS

LtHandle(in)

Handle to LT-XXX obtained from **LT_OpenDevice()**.

RETURN VALUE

Returns zero on success. A return value < 0 indicates an error.

COMMENTS

This function closes the connection to the LT-XXX device and free all allocated resources. After this call *LtHandle* is invalid and can no longer be used.

USAGE EXAMPLE

```
LT_CloseDevice( LtHandle );
```

Configuration functions

LT_SETPARAM ()

This function sets one video or encoder parameter. The counterpart of this function is LT_GetParam which retrieves internally recalculated or read-only values. The LT_GetParam() function will be described in a dedicated chapter.

```
int LT_SetParam(    int LtHandle,  
                   int ParamKey,  
                   int ParamValue );
```

ARGUMENTS

LtHandle(in)
Handle to LT-XXX obtained from LT_OpenDevice ()

ParamKey (in)
Index number to parameter

ParamValue (in)
New parameter value

RETURN VALUE

Returns zero on success. A return value < 0 indicates an error.

COMMENTS

This function only sets parameter values stored within internal data structures. For parameter changes to actually take effect they must be uploaded to the LT-XXX hardware by calling LT_UploadDeviceParams().

For parameters like video input resolution (LT_PARAMS_VIDEO_RESOLUTION), two modes are possible: automatic and manual. For manual mode, user must select video resolution using a macro of e.g type "V480I" as parameter value. For automatic mode, video input resolution and framerate is detected automatically. The examples bellow, show how the different modes are set.

NOTE: The "AUTO" keyword can only be set for video resolution and not yet for video input or colorspace.

NOTE: Using this function a colorbar test mode can be enabled where the video inputs are not used (LT_SetParam(LtHandle,LT_PARAMS_VIDEO_INPUT,VID_COLORBAR);).

HINT: Use the colorbar test mode to verify that the hardware and software are functioning correctly independently from any real video source.

USAGE EXAMPLE

This example shows how to setup LT-XXX for automatic video input detection.

```
// Reset all parameters to their default values
LT_SetParam(LtHandle, LT_PARAMS_SET_DEFAULT, 1);

// DVI-D Input Connector
LT_SetParam( LtHandle, LT_PARAMS_VIDEO_INPUT, VID_HD_DIGITAL);
// Input Resolution
LT_SetParam( LtHandle, LT_PARAMS_VIDEO_RESOLUTION, AUTO);
// Input Colorspace
LT_SetParam( LtHandle, LT_PARAMS_VIDEO_COLORSPACE, VID_RGB);
```

This example shows how to setup LT-XXX for manual mode.

```
// DVI-D Input Connector
LT_SetParam(LtHandle,LT_PARAMS_VIDEO_INPUT,VID_HD_DIGITAL);
// Input Resolution
LT_SetParam(LtHandle,LT_PARAMS_VIDEO_RESOLUTION,V1080P30);
// Input Colorspace
LT_SetParam(LtHandle,LT_PARAMS_VIDEO_COLORSPACE,VID_RGB);
```

VALID PARAMETER INDICES AND VALUES FOR LT_SetPARAM() AND LT_GetPARAM()

Initialisation

Reset all configuration parameters to their default values through setting:

```
LT_SetParam(LtHandle, LT_PARAMS_SET_DEFAULT, 1);
```

<u>Key</u>	<u>Value</u>
LT_PARAMS_SET_DEFAULT	1 : force reset of configuration parameters

Video input connection type

Selects the input video source

<u>Key</u>	<u>Value</u>
LT_PARAMS_VIDEO_INPUT	VID_COLORBAR
	VID_COPROCESSING
	VID_BOOTSCREEN ¹
	VID_SDI
	VID_SDI1 (same than VID_SDI)
	VID_SDI2 ²
	VID_HD_AUTO

VID_HD_DIGITAL
 VID_HD_DIGITAL1(same than VID_HD_DIGITAL)
 VID_HD_DIGITAL2³
 VID_HD_ANALOG
 VID_HD_ANALOG1(same than VID_HD_ANALOG)
 VID_HD_ANALOG2⁴
 VID_COMPOSITE
 VID_COMPOSITE1(same than VID_COMPOSITE)
 VID_COMPOSITE2⁵
 VID_SVIDEO

¹Only valid for custom board

²Only valid for LT-122

³Only valid for LT-124 and LT-125

⁴ Only valid for custom board

⁵ Only valid for custom board

Video input signal type

<u>Key</u>	<u>Value</u>
LT_PARAMS_VIDEO_RESOLUTION	AUTO (default)
	VCUSTOM0-31
	VESA640X400_85HZ
	VESA720X400_85HZ
	IS640X480_60HZ
	VESA640X480_72HZ
	VESA640X480_75HZ
	VESA640X480_85HZ
	VESA800X600_56HZ
	VESA800X600_60HZ
	VESA800X600_72HZ
	VESA800X600_75HZ
	VESA800X600_85HZ
	VESA848X480_60HZ
	VESA1024X768_60HZ
	VESA1024X768_70HZ
	VESA1024X768_75HZ
	VESA1024X768_85HZ
	VESA1152X864_75HZ
	CVTRB1280X768_60HZ
	CVT1280X768_60HZ
	CVT1280X768_75HZ
	CVT1280X768_85HZ
	VESA1280X960_60HZ
	VESA1280X960_85HZ
	VESA1280X1024_60HZ
	VESA1280X1024_75HZ

VESA1360X768_60HZ
CVTRB1400X1050_60HZ
CVT1400X1050_60HZ
CVTRB1440X900_60HZ
CVT1440X900_60HZ
CVT1440X900_75HZ
CVTRB1600X1200_60HZ
VESA1600X1200_60HZ
CVTRB1680X1050_60HZ
CVT1680X1050_60HZ
CVTRB1920X1200_60HZ
GTF720X480_60HZ
CVT720X480_60HZ
GTF720X576_50HZ
CVT720X576_50HZ
GTF960X600_60HZ
CVT960X600_60HZ
GTF1024X768_60HZ
CVT1024X768_60HZ
GTF1280X720_60HZ
CVT1280X720_60HZ
GTF1280X1024_60HZ
CVT1280X1024_60HZ
VESA1152X768_55HZ
GTF1152X864_60HZ
CVT1152X864_60HZ
VESA1152X864_85HZ
GTF1280X768_60HZ
GTF1280X800_60HZ
CVT1280X800_60HZ
CVTRB1280X800_60HZ
VESA1280X1024_85HZ
CVT1600X1024_60HZ
CVTRB1920X1080_60HZ
CVTRB1920X1080_50HZ
CVT1920X1080_50HZ
V480I
V480IDVI
V576I
V576I100
V576IDVI
V480P30
V480P2997
V480P2997DVI
V480P60
V480P5994
V576P25
V576P25DVI
V576P50
V720P60
V720P5994
V720P50
V720P30
V720P2997

V720P25
V720P24
V720P2398
V1080I50a
V1080P50a
V1035I60
V1035I5994
V1080P60
V1080P5994
V1080P50b
V1080I60
V1080I5994
V1080I50b
V1080P30
V1080P2997
V1080P25
V1080P24
V1080P2398
V720P24RB
V720P2398RB
SQUAREPAL
V576I100SQ
HICOR1168X1168_50HZ
HICOR1024X944_60HZ
INNOVA1280X1024_72HZ
OEC1024X1024_75HZ

<u>Key</u>	<u>Value</u>
LT_PARAMS_VIDEO_SQUAREPAL_FLAG	0 (default) Treat Pal as not squared
	1 Treat Pal as squared

Video input signal color space

Selects the input video color encoding. This only applies on to signals input on the DVI-I connector (DVI/RGB/YUV). SDI, Composite and S-video signals do not allow for alternative color spaces.

<u>Key</u>	<u>Value</u>
LT_PARAMS_VIDEO_COLORSPACE	VID_RGB (default) VID_YUV

Still image capture time out in unit of fetch cycles

Amount of fetch cycles after which still capture returns an error.

<u>Key</u>	<u>Value</u>
LT_PARAMS_STILL_TIMEOUT	0: (default)

VIDEO ANALOG PARAMETER manual mode selection

Manual fine-tuning of ADV9880 RGB values in case of DVI-A signal. If set this value allows manual setting of an offset and a gain on RGB components via:

- LT_PARAMS_VIDEO_ANALOG_OFFSET_RED : default: 128, 0 low offset, 255 large offset
- LT_PARAMS_VIDEO_ANALOG_OFFSET_GREEN : default: 128, 0 low offset, 255 large offset
- LT_PARAMS_VIDEO_ANALOG_OFFSET_BLUE : default: 128, 0 low offset, 255 large offset
- LT_PARAMS_VIDEO_ANALOG_GAIN_RED : default: 128, 0 low gain, 255 large gain
- LT_PARAMS_VIDEO_ANALOG_GAIN_GREEN : default: 128, 0 low gain, 255 large gain
- LT_PARAMS_VIDEO_ANALOG_GAIN_BLUE : default: 128, 0 low gain, 255 large gain

<u>Key</u>	<u>Value</u>
LT_PARAMS_VIDEO_ANALOG_OFFSET_MANUAL	AUTO: (default) 1: ON

Video input signal color shift

If set, this parameter inverts U and V chroma components in YUV4:2:2 incoming video.

An AD converter chip is used to receive input video right from the DVI-I connector. This is for example Analog Devices ADV9880 in case of the LT-102 and LT-101. Prior to video acquisition and compression an RGB to YUV 4:2:2 color space conversion is done in the AD chip. The FOURCC used here is YUYV. When set, this parameter switches the U and the V components in the above FOURCC. This is e.g used to compensate for a color inversion issue observed in the now obsolete ADV9880 that prevents video from rare cameras (e.g Storz) to be sampled correctly. There is no way to automatically detect in which case this fix should be applied.

This parameter may also be used to achieve special visual effects.

Boards affected: LT-101, LT-102

<u>Key</u>	<u>Value</u>
LT_PARAMS_VIDEO_COLOR_SHIFT	0: (default) 1: ON

Use internal DVI data enable

If set this parameter ignores inbuilt DVI data enable (DE) signal embedded in the DVI video in favor of a hardware internal computed DE signal. This parameter should only be used in a debug situation to assess that a poorly synchronized DVI video is present. It has been observed that certain video sources provide poorly synchronized DVI signal that cannot be correctly converted with ADV9880 chips. This appears to be a limitation of the now obsolete ADV9880 being less tolerant than other chips against weak DVI signals.

Boards affected: LT-101, LT-102

<u>Key</u>	<u>Value</u>
LT_PARAMS_VIDEO_USE_INTERNAL_DVI_DE	0 : OFF (default) 1 : ON

Denoise filter enable

Enables the video low pass filter. The filter is at the compression input.

<u>Key</u>	<u>Value</u>
LT_PARAMS_VIDEO_DENOISE_FILTER	0 : OFF (default) 1 : LOW 2 : MEDIUM 3 : STRONG

Video input shift

These keys allow for the shifting of the image position within the active window.

<u>Key</u>	<u>Value</u>	
LT_PARAMS_VIDEO_VSHIFT	0 to 2048 (default: 0)	Vertical shift
LT_PARAMS_VIDEO_HSHIFT	0 to 2048 (default: 0)	Horizontal shift

Video cropping

These parameters are rounded to the best valid values. CROP_HSTART + CROP_WIDTH and CROP_VSTART + CROP_HEIGHT should not respectively exceed video width and video height.

<u>Key</u>	<u>Value</u>
LT_PARAMS_CROP_HSTART	(default: 0) Start from left border
LT_PARAMS_CROP_VSTART	(default: 0) Start from top border
LT_PARAMS_CROP_WIDTH	(default: 0) Active video width
LT_PARAMS_CROP_HEIGHT	(default: 0) Active video height

Video Hue Saturation Brightness Contrast

These keys are used for color improvement.

<u>Key</u>	<u>Value</u>
LT_PARAMS_VIDEO_HUE	-30 to 30 (default: 0)
LT_PARAMS_VIDEO_SATURATION	0 to 255 (default: 128)
LT_PARAMS_VIDEO_BRIGHTNESS	-128 to 127 (default: 0)

LT_PARAMS_VIDEO_CONTRAST	0 to 255 (default: 128)
--------------------------	-------------------------

Input video resolution

These parameters are read-only parameters to be fetched through `LT_GetParam()` function. they specify input video resolution if no downscaling or cropping is used. If downscaling or cropping is used, these parameters specify video resolution after the processing.

<u>Key</u>	<u>Value</u>
LT_PARAMS_VIDEO_WIDTH	1 to 2048 (default: 1920)
LT_PARAMS_VIDEO_HEIGHT	1 to 2048 (default: 1088)

Target bitrate

Target compressed video bitrate. Expressed in bits/sec.

<u>Key</u>	<u>Value</u>
LT_PARAMS_BRC_BITRATE	32000 to 80000000 (default: 10000000)

Bitrate control mode

BitRate Control (BRC) mode will decide which policy should be applied in order to achieve target bitrate.

<u>Key</u>	<u>Value</u>
LT_PARAMS_BRC_MODE	<p>BRC_ABR_WIDE For recording application. There can be big variation around the targeted bitrate.</p> <p>BRC_ABR_MEDIUM For recording application. There will be medium variation around the targeted bitrate.</p> <p>BRC_VBR (default) For networking application. The maximum bitrate should not exceed target bitrate by more than 10%, unless target bitrate is too low. Forced skip frame can occur. (Skip frame feature is not yet implemented for H.264)</p> <p>BRC_CBR (default) Near CBR behavior.</p> <p>BRC_CONSTANTQ For recording application with constant quality. Bitrate target is not taken in account here.</p> <p>BRC_CUSTOM If this parameter is set, user can select custom BRC parameters. Please refer to Custom BRC Parameter section</p>

Video quality

Video quality can be constrained to have minimum and maximum value. This factor will directly set VC-1 / H.264 picture quantizer.

<u>Key</u>	<u>Value</u>
LT_PARAMS_BRC_QUALITY_MIN	<p>0 to 31 for VC-1 (default: AUTO=0)</p> <p>0 to 51 for H.264 (default: AUTO=0)</p> <p>Define best quality factor. Lower is better.</p> <p>Usually quality above 2 is set.</p>
LT_PARAMS_BRC_QUALITY_MAX	0 to 31 (default: AUTO=0)

0 to 51 for H.264 (default: AUTO=0)
 Define worse quality factor. Higher is better.
 Usually quality around 20 is set.

Video quality jump

Is the offset between Intra frame and Inter frame picture-quantizer. It is not recommended to change the default configuration.

<u>Key</u>	<u>Value</u>
LT_PARAMS_BRC_QUALITY_JUMP	-7 to 7 (default: AUTO=0)

GOP length

Sets the Group Of Pictures (GOP) length. A GOP is made up of one I-frame followed by zero or more P-frames.

<u>Key</u>	<u>Value</u>	
LT_PARAMS_BRC_GOP_LENGTH	1	I-frame only structure
	2 to 4095(default: 120)	IP structure. The number of P-frames in GOP is this value minus 1

Intra SAD threshhold

The P-frame intra macroblock SAD threshold steers the number of intra coded macroblocks within an image. This parameter is only used in VC-1.

<u>Key</u>	<u>Value</u>	
LT_PARAMS_ENC_INTRA_SAD_THRESHOLD	0	All macroblocks set to "intra mode".
	1 to 65535 (default: 2500)	

Range map

Range map parameter as specified in VC-1 decoding standard (SMPTE 421M-2006).

<u>Key</u>	<u>Value</u>
LT_PARAMS_ENC_RANGEMAP_LUMA	0 to 3 (default: -1)
LT_PARAMS_ENC_RANGEMAP_CHROMA	0 to 3 (default: -1)

Target frame rate

Target output framerate. If this value is lower than the video input framerate, framerate decimation will be performed. This value cannot be higher than the actual input video framerate. LT-XXX will try to achieve output target framerate but the actual framerate may vary slightly from target. If set to AUTO, no decimation will occur unless the frame rate resolution combination exceeds the computational capacity of the LT-xxx. In this case the frame rate will be decimated.

<u>Key</u>	<u>Value</u>
LT_PARAMS_ENC_FRAMERATE	1 to input frame rate AUTO (default)

Uniform and non-uniform quatization

Selects between uniform and non-uniform quantization according to VC-1 specification (SMPTE 421M-2006). If set to 0, an automatic selection will be performed.

<u>Key</u>	<u>Value</u>
LT_PARAMS_ENC_QUALITY_UNIFORM	0 to 2 (default: 0)

- 1: Uniform
- 2: Non-uniform

Custom BRC parameters

Some selected parameters can be directly set by user in order to steer BRC behavior manually.

<u>Key</u>	<u>Value</u>
LT_PARAMS_BRC_QUALITY_SPEED_UP	1 to 256: Adjusts the BRC algorithm reaction speed to scene changes. lower is fastest
LT_PARAMS_BRC_QUALITY_SPEED_DOWN	1 to 256: Adjusts the BRC algorithm reaction speed to scene changes. lower is fastest
LT_PARAMS_BRC_VB_AVG_WINDOW	0 to 15: Bitrate control Buffer sliding window size. This affects the reaction time of the bitrate control algorithm. This value is expressed as power of two.
LT_PARAMS_BRC_VB_DELTAMIN	-32768 to +32767: Virtual buffer control parameter
LT_PARAMS_BRC_VB_DELTAMAX	-32768 to +32767: Virtual buffer control parameter
LT_PARAMS_BRC_VB_DELTASKIP	-32768 to +32767: Virtual buffer control parameter that decides when to insert skip frames into the VC-1 stream

Video downscaling

Enables and programs generic video downscaling parameters for the compressed video. Note that only values below original values can be set. No upscaling is feasible yet.

<u>Key</u>	<u>Value</u>
LT_PARAMS_ENC_WIDTH	0 (default) No downscaling Up to input video width. A maximum value of 128 is accepted. Need to be a multiple of 16 Parameter is rounded to nearest valid value
LT_PARAMS_ENC_HEIGHT	0 (default) No downscaling Up to input video height. A maximum value of 128 is accepted. Need to be a multiple of 16. Parameter is rounded to nearest valid value

Selects audio Input

Selects audio source.

<u>Key</u>	<u>Value</u>
LT_PARAMS_AUDIO_INPUT	AUD_ANALOG (default) Audio from stereo analog source (Audio jack) AUD_ANALOG1 (same input than AUD_ANALOG) Audio from stereo analog source (Audio jack) AUD_ANALOG2 (Only available on LT-122) Audio from second stereo analog source (Audio jack) AUD_EMBEDDED Audio embedded in video stream (HDMI, SDI or DVI-D)

Sets audio mode

Sets audio acquisition and compression mode.

<u>Key</u>	<u>Value</u>
LT_PARAMS_AUDIO_CODEC	AUD_NONE (default)
	AUD_PCM_16BIT_STEREO_48KHZ PCM raw stereo 16bit/channel audio sampled at 48KHz
	AUD_WMA8_STEREO_48KHZ Windows media audio 8. Encoding is done in host CPU
	AUD_AAC_STEREO_48KHZ AAC audio. Encoding is done in host CPU

Audio bitrate

Sets target audio bitrate for compressed audio modes (WMA8 or AAC). Expressed in bits/sec.

<u>Key</u>	<u>Value</u>
LT_PARAMS_AUDIO_BITRATE	64000 to 1536000 (default: 192000)

Audio volume

Valy the volume between 0 and 12dB

NOTE: The volume register is only set if LT_PARAMS_AUDIO_VOLUME is used.

<u>Key</u>	<u>Value</u>
LT_PARAMS_AUDIO_VOLUME	0 to 12

Raw/uncompressed scaling

Enables and programs generic video downscaling parameters for the uncompressed video. Note that only values below original values can be set. No upscaling is feasible yet.

<u>Key</u>	<u>Value</u>	
LT_PARAMS_RAW_WIDTH	0 (default)	No downscaling Up to input video width. A maximum value of 64 is accepted. Need to be a multiple of 16 Parameter is rounded to nearest valid value
LT_PARAMS_RAW_HEIGHT	0 (default)	No downscaling Up to input video height. A maximum value of 64 is accepted. Need to be a multiple of 16. Parameter is rounded to nearest valid value

Enable/Desable deinterlacing

Activates or deactivates the deinterlacing filter.

<u>Key</u>	<u>Value</u>
LT_PARAMS_VIDEO_DEINTELACING_FILTER	OFF ON AUTO (default)

Encoder profile

H.264 profile indication

For VC-1 encoder advanced profile is always used

<u>Key</u>	<u>Value</u>
LT_PARAMS_ENC_BITSTREAM_PROFILE	0: H.264 Baseline 1: H.264 Main 3: H.264 Advanced (default)

Encoder level

VC-1/H.264 level indication

For VC-1 encoder following coding is used:

<u>Key</u>	<u>Value</u>
LT_PARAMS_ENC_BITSTREAM_LEVEL	(default: 3) 0: Advanced Profile Level Level 0 1: Advanced Profile Level Level 1 2: Advanced Profile Level Level 2 3: Advanced Profile Level Level 3

For H.264 encoder following coding is used (this value will be divided by 10 in order to get actual level. E.g 42 is level 4.2):

<u>Key</u>	<u>Value</u>
LT_PARAMS_ENC_BITSTREAM_LEVEL	0 to 42 (default: 40)

VC-1 entry point layer enable

It is possible to have an entry point code inserted before each I-frame in the VC-1 stream. This allows for easy indexing of the produced bitstream.

Note: Whatever this setting is, an entry point layer syntax element will always be inserted before the first I-frame of any stream.

<u>Key</u>	<u>Value</u>	
LT_PARAMS_ENC_EPL_ALWAYS_PRESENT	0	don't use entry point layer syntax element before every I-frame.
	1	Use entry point layer syntax element before each I-frame

Frame rate information

This is an integer offset in Part Per Million (PPM) to the video input frame interval (1/framerate) information. The frame interval is calculated internally and an offset can be added to it before retrieving in order to fine-tune the video timestamps.

Note 1: Please also refer to LT_GetHighPrecFrInterval() function that shows how to retrieve the frame interval value.

Note 2: A timestamp directly derived from the boards internal reference clock can also be gathered through "LT_META_***" parameters as shown later.

<u>Key</u>	<u>Value</u>
LT_PARAMS_PPM_FR_INTERVAL_OFFSET	(default: 0)

Mpeg Transport Stream multiplexer enabling

This is a flag that enables MPEG-TS multiplexing. This flag has to be set if the user wants to use TS multiplexing feature.

<u>Key</u>	<u>Value</u>
LT_PARAMS_TS_ENABLE_TS	0 (default) 1

Mpeg Transport Stream mode

This is a flag that selects between MPEG-TS and M2TS

<u>Key</u>	<u>Value</u>
LT_PARAMS_TS_USE_M2TS	0 (default) 1

Mpeg Transport Stream PA PMT interval

This parameter steers the insertion interval of MPEG-TS PA PMT section and is expressed in seconds.

<u>Key</u>	<u>Value</u>
LT_PARAMS_TS_PA_PMT_INT	0 to $2^{32}-1$ (default: 0)

ASF multiplexer enabling

This is a flag that enables ASF multiplexing. This flag has to be set if the user wants to use ASF multiplexing feature.

<u>Key</u>	<u>Value</u>
LT_PARAMS_ASF_ENABLE_ASF	0 (default) 1

ASF packet size

This is the length of ASF packets. The ASF header may have a different length than the one specified in LT_PARAMS_ASF_PACKET_SIZE. If the ASF header should have same length than the packets (or a multiple of LT_PARAMS_ASF_PACKET_SIZE) then LT_PARAMS_ASF_USE_PADDING must be set to 1.

<u>Key</u>	<u>Value</u>
LT_PARAMS_ASF_PACKET_SIZE	(default: 2048)

ASF preroll

This is an initial ASF display delay that is expressed in milliseconds.

<u>Key</u>	<u>Value</u>
LT_PARAMS_ASF_PREROLL	0 to $2^{32}-1$ (default: 33)

ASF padding

Use padding for ASF header to have same length than ASF packets.

<u>Key</u>	<u>Value</u>	
LT_PARAMS_ASF_USE_PADDING	0 (default)	Not used

Hotplug Audio/Video input cable

This sets the time-out of audio/video data acquisition in millisecond to allow for recovery after unplugging of a data cable. If set to 0, middleware will wait forever for audio/video data (after calling LT_GetEncFrame() or similar functions) and no error code is returned to user.

If set to 1000, middleware will wait for one second for audio/video data (after calling LT_GetEncFrame() or similar functions) and returns an error code.

<u>Key</u>	<u>Value</u>	
LT_PARAMS_ENC_TIMEOUT	1000 (default)	One second
	-1	Unlimited
	0 to 2 ³¹ -1	Timeout in milliseconds
LT_PARAMS_RAW_TIMEOUT	1000 (default)	One second
	-1	Unlimited
	0 to 2 ³¹ -1	Timeout in milliseconds
LT_PARAMS_AUDIO_TIMEOUT	1000 (default)	One second
	-1	Unlimited
	0 to 2 ³¹ -1	Timeout in milliseconds

Modulo 16 rounding of frame height

Some video resolutions like 1920x1080 cannot be processed as such because they are not a multiple of 16x16pixels (1080/16 is not an integer). Only multiple of 16 sized images can be compressed (this is called macroblock partitioning in video compression jargon). One can do either black line/column insertion or remove 8 lines/column to accommodate for this. In the case of 1080, a rounding can be done towards 1072 or towards 1088.

<u>Key</u>	<u>Value</u>	
LT_PARAMS_VIDEO_16RESMODE	0 (default)	round down
	1	round up

Interlaced to progressive

This will double the framerate and perform interpolation for each field.

<u>Key</u>	<u>Value</u>
LT_PARAMS_VIDEO_INTERLACED_TO_PROGRESSIVE	OFF
	ON
	AUTO (default)

Field polarity

If set, top field will be transmitted first, otherwise bottom field is transmitted first.

<u>Key</u>	<u>Value</u>
LT_PARAMS_VIDEO_FIELDPOLARITY	0: OFF (default)
	1: ON

Use field as frame

Uses each interlaced field as a full frame. Image resolution is reduced but temporal resolution is increased

<u>Key</u>	<u>Value</u>
LT_PARAMS_VIDEO_USEFIELDASFRAME	OFF
	ON
	AUTO (default)

Deinterlacing strategy

Switch between a simple and a more advanced deinterlacing algorithms

<u>Key</u>	<u>Value</u>
LT_PARAMS_VIDEO_DEINTELACING_STRATEGY	0 Median based (default) 1 Median and simple Motion Estimation mix

Panic thresholds

These parameters set the panic threshold in terms of percentage of input buffer fullness. If achieved, input video/audio samples will be discarded at acquisition.

<u>Key</u>	<u>Value</u>
LT_PARAMS_ENC_BUFFER_PANIKTHR	0 to 100% (75: default)
LT_PARAMS_AUDIO_BUFFER_PANIKTHR	0 to 100% (75: default)

Use squared pixels

Use PAL squared pixels.

<u>Key</u>	<u>Value</u>
LT_PARAMS_VIDEO_SQUAREPAL_FLAG	OFF ON AUTO (default)

Read only info parameters

These parameters are read only (gathered though LT_GetParam() call). They will provide useful information about hardware status and software versions.

<u>Key</u>	<u>Value</u>
LT_INFO_TRANSFER_BITRATE	integer. Actual Instantaneous bitrate (in bits per second)
LT_INFO_GOP_BITRATE	integer. Average bitrate over one GOP
LT_INFO_ENC_BUFFER_LEVEL	compressed video internal cache. 0 to 2047 memory lines
LT_INFO_AUDIO_BUFFER_LEVEL	Audio internal cache. 0 to 512 memory lines
LT_INFO_RAW_BUFFER_LEVEL	Raw video internal cache. 0 to 4095 memory lines
LT_INFO_DROPPED_FRAMES	integer. Amount of dropped frames since the device has been started
LT_INFO_DLL_VERSION	integer. Middleware version
LT_INFO_BOARD_TEMPERATURE	integer. Board temperature. Not available on LT101 and LT102 series
LT_INFO_BOARD_SERIAL_NUMBER	integer. Board unique serial number (also printed on the board)
LT_INFO_DEVICE_ID	integer. Board's unique ID: LT101: 0x1010 LT102: 0x1020 LT122: 0x1220 LT124: 0x1240 LT125: 0x1250 LT200: 0x2000
LT_INFO_DEVICE_INTERFACE	integer. Device interface type: USB: 1 PCI: 2 PCIE: 3
LT_INFO_ENC_VIDEOCODEC	integer. Device interface type: NULL: 0 VC1: 1 H264: 2

AAC audio encoding

These keys set AAC audio encoding parameters and are only valid if AAC audio encoding is enabled through setting of:

LT_SetParam(LtHandle,LT_PARAMS_AUDIO_CODEC,AUD_AAC_STEREO_48KHZ);

Note: AAC encoding is done in software through the host computer. The processing power needed to encode audio in AAC is negligible.

<u>Key</u>	<u>Value</u>
LT_PARAMS_AUD_AAC_MPEG_VERSION	0: MPEG2 (default) 1: MPEG4
LT_PARAMS_AUD_AAC_OBJECT_TYPE	AAC object type 1: MAIN 2: LOW (default) 3: SSR (Scaleable Sampling Rate) 4: LTP (Long Term Prediction)
LT_PARAMS_AUD_AAC_ALLOW_MID_SIDE	Allow mid/side coding 1: allow 0: disable (default)
LT_PARAMS_AUD_AAC_USE_LFE	Use one of the channels as LFE channel 1: allow 0: disable (default)
LT_PARAMS_AUD_AAC_USE_TNS	Use Temporal Noise Shaping 1: allow 0: disable (default)
LT_PARAMS_AUD_AAC_BANDWIDTH	AAC file frequency bandwidth 16000 (default)
LT_PARAMS_AUD_AAC_QUANT_QUAL	Quantizer quality 500 (default)
LT_PARAMS_AUD_AAC_OUTPUT_FORMAT	Bitstream output format 0: Raw 1: ADTS (default)

Use internal DVI data enable flag

Use internal DVI-D data enable flag instead of the one provided by A/D chip

<u>Key</u>	<u>Value</u>
LT_PARAMS_VIDEO_USE_INTERNAL_DVI_DE	0 (default) 1

Select compression channel (not available for LT101/LT102)

Select one out of two compression channels

<u>Key</u>	<u>Value</u>
LT_PARAMS_CHANNEL_SELECT	0 (default) 1

Select resource type

Select the resource you want to access on the selected channel (channel selection is done through LT_PARAMS_CHANNEL_SELECT). This feature is useful if the user wants to use the API library in different applications or processes. A user can for example select channel type audio in one process and raw (uncompressed video) in another process.

Note: User can also "OR" between different values like this:

LT_SetParam(LtHandle, LT_PARAMS_CHANNEL_TYPE, CHANNEL_TYPE_RAW | CHANNEL_TYPE_AUD);

<u>Key</u>	<u>Value</u>
LT_PARAMS_CHANNEL_TYPE	CHANNEL_TYPE_NULL nothing to do CHANNEL_TYPE_ENC VC-1 or H.264 video

CHANNEL_TYPE_RAW uncompressed video
CHANNEL_TYPE_AUD audio
CHANNEL_TYPE_STILL still image
CHANNEL_TYPE_ALL all data types (default)

Select Firmware type

Select the type of compression to be applied. For a given codec (VC-1 or H.264) , 3 types of firmware are available:

1. SD: This is a special firmware for standard video signal
2. P30: If this firmware is used, the board (LT-122, LT-124, LT-200 or LT-125) can compress two encoding channels in parallel.
3. P60: If this firmware is used, the board (LT-122, LT-124, LT-200 or LT-125) can compress only one encoding channels up to 60fps.

The amount of firmware available for a given board is determined by the amount of *.bit files in the install directory of the driver. If LT101 has been installed for example, lt101.bit and lt101_h264.bit firmware files will be present in the installation directory representing VC-1 compression capability (lt101.bit) and H.264 compression capability (lt101_h264.bit).

Example:

Select VC-1 compression type for LT101/102:

Nothing to do. This will be done by default

Select H.264 compression type for LT101/102:

LT_SetParam(LtHandle,LT_PARAMS_FW_SELECT,FW_H264_P30);

<u>Key</u>	<u>Value</u>
LT_PARAMS_FW_SELECT	FW_VC1_SD
	FW_VC1_P30 (default)
	FW_VC1_P60
	FW_H264_SD
	FW_H264_P30
	FW_H264_P60

Use black and white detection mode on standard video input

Allows for black and white input signal types.

<u>Key</u>	<u>Value</u>
LT_PARAMS_SD_BW_ACQUISITION	0: off (default)
	1: on

Metadata transport over video transmission channel

Some metadata are always transferred during compressed video acquisition as user data and discarded prior to transmission through LT_GetEncFrame() function. In order to access metadata on uncompressed channel, user need to ask for it explicitly (through LT_PARAMS_RAW_METADATA key) since metadata is inserted directly at the end of each picture in the payload data. The various data available in the metadata structure are shown as follows:

NOTE: all parameters are read only except LT_PARAMS_RAW_METADATA

NOTE: LT_META_RAW_*** designates metadata transferred with uncompresses video and

LT_META_ENC_*** designates metadata transferred with compresses video

NOTE: Time stamps are incremented in nanoseconds units. They are contained in a 64bit variable transmitted in 32bit high part and 32bit low part: $ts = ((ts_high \ll 32) + ts_low)$

<u>Key</u>	<u>Value</u>
LT_PARAMS_RAW_METADATA	0: do not insert metadata to Raw Video (default) 1: insert metadata to Raw Video
LT_META_RAW_FRAME_FID	Field ID 0: top field if LT_PARAMS_VIDEO_FIELDPOLARITY == 0 1: bottom field in above case
LT_META_ENC_FRAME_FID	Field ID 0: top field if LT_PARAMS_VIDEO_FIELDPOLARITY == 0 1: bottom field in above case
LT_META_RAW_FRAME_TS_LO	low part of hardware time stamp in nanoseconds 32bit Integer
LT_META_ENC_FRAME_TS_LO	low part of hardware time stamp in nanoseconds 32bit Integer
LT_META_RAW_FRAME_TS_HI	high part of hardware time stamp in nanoseconds 32bit Integer
LT_META_ENC_FRAME_TS_HI	high part of hardware time stamp in nanoseconds 32bit Integer

LT_GETPARAM ()

This function retrieves some internally determined parameters from the API.

```
int LT_GetParam(    int LtHandle,
                   int ParamKey,
                   int * pParamValue );
```

ARGUMENTS

LtHandle(in)
Handle to LT-XXX obtained from LT_OpenDevice ()

ParamKey (in)
Index number to parameter

pParamValue (out)
Pointer to parameter value

RETURN VALUE

Returns zero on success. A return value < 0 indicates an error.

COMMENTS

This function reads parameter values stored within internal data structures. Unless `LT_UploadDeviceParams()` has been called the values read may not reflect the values actually being used by the hardware. See `LT_SetParam()` for valid index values.

Use this function to get the actual parameters to be used by LT-XXX hardware as they may be modified by `LT_SetParam()`. For example, if scaling is applied to the encoded video input pipeline you can use `LT_GetParam()` to obtain the actual scaled resolution. Usually this function is used after `LT_UploadDeviceParameters()` to retrieve width and height of output.

USAGE EXAMPLE

```
Int Status = 0;
Int LtHandle = 0;
Double frame_interval;
Double frame_rate;
Int enc_width;
Int enc_height;
Int raw_width;
Int raw_height;

Status = LT_UploadDeviceParameters(LtHandle);

if( Status < 0 )
{
    printf("LT_UploadDeviceParameters error : %d\n",Status);
    return Status;
}

// At this point LT-XXX device is ready to be used
LT_GetHighPrecFrInterval(LtHandle, &frame_interval, &frame_rate); //frame_interval in
millisecond units

//get some params back from SDK
LT_GetParam(LtHandle, LT_PARAMS_ENC_WIDTH, &enc_width);
LT_GetParam(LtHandle, LT_PARAMS_ENC_HEIGHT, &enc_height);
LT_GetParam(LtHandle, LT_PARAMS_RAW_WIDTH, &raw_width);
LT_GetParam(LtHandle, LT_PARAMS_RAW_HEIGHT, &raw_height);
```


LT_UPLOADDEVICEPARAMETERS()

This function uploads all configurations parameters to the LT-XXX hardware. It also uploads the *.bit firmware file (also sometimes referred as FPGA bitstream) to the device if this has not been done yet. While the device is powered, subsequent calls to LT_UploadDeviceParameters() will not imply upload of the firmware again. The bitstream is uploaded only ones. This mechanism has been implemented to shorten the upload time that usually takes less than one second.

```
int LT_UploadDeviceParameters ( int * LtHandle );
```

ARGUMENTS

LtHandle(in)

Handle to LT-XXX obtained from [LT_OpenDevice\(\)](#).

RETURN VALUE

Returns zero on success. A return value < 0 indicates an error.

COMMENTS

This function uploads parameter values stored within internal data structures to the LT-XXX hardware and must be called at least once prior to an LT_StartEnc() or LT_StartRawCapture() call.

In the upload process many internal data structure elements are recalculated during a consistency check and may no longer correspond to the exact parameter values initially set. Use functions such as LT_GetParam(), LT_GetHighPrecFrInterval(), etc., to obtain the actual values used.

Prior to a LT_UploadDeviceParameters() call, the internal data structure elements can be modified using LT_SetParam().

USAGE EXAMPLE

```
int Status = 0;
Status = LT_UploadDeviceParameters(LtHandle);
if( Status < 0 )
    return -1;
```

LT_GETHIGHPRECFRINTERVAL()

This function gets high (double) precision frame interval (in millisecond units) and framerate (frames/sec).

```
int LT_GetHighPrecFrInterval(    int LtHandle,  
                                double *pHighPrecFrInterval,  
                                double *pHighPrecFrRate );
```

ARGUMENTS

LtHandle(in)

Handle to LT-XXX obtained from LT_OpenDevice ()

pHighPrecFrInterval(out)

High precision frame interval measured in millisecond units.

pHighPrecFrRate(out)

High precision frame rate measured frames/sec.

RETURN VALUE

Returns zero on success. A return value < 0 indicates an error.

COMMENTS

Note that this function can only be called after LT_UploadDeviceParameters(). The reason for that is that the user can reset the target framerate using LT_SetParam() to another value using the LT_PARAMS_ENC_FRAMERATE parameter.

If the framerate chosen by user cannot be met precisely an internal approximation will be computed. This internally computed framerate approximation and corresponding frame interval is reported through LT_GetHighPrecFrInterval().

LT_SetParam() function and LT_PARAMS_ENC_FRAMERATE parameter have been described earlier in this document.

User can add a positive or a negative offset (in PPM) to the frame interval in order to e.g steer/adjust lip-sync.

Also note that hardware timestamps using internal clock reference (in nanosecond ticks) can be obtained together with each video frame using LT_META_RAW_FRAME_TS_LO, LT_META_RAW_FRAME_TS_HI, LT_META_ENC_FRAME_TS_LO and LT_META_ENC_FRAME_TS_HI as describe above.

USAGE EXAMPLE

```
double frame_interval;  
double frame_rate;  
int target_framerate = 15;  
  
LT_SetParam(LtHandle,LT_PARAMS_ENC_FRAMERATE,target_framerate);  
LT_SetParam(LtHandle, LT_PARAMS_PPM_FR_INTERVAL_OFFSET, 1000);  
  
...  
  
LT_UploadDeviceParameters(LtHandle);  
LT_GetHighPrecFrInterval(LtHandle, &frame_interval, &frame_rate);
```

LT_GETVIDEODESCRIPTION()

This function retrieves a long or a short description of the video signal.

```
int LT_GetVideoDescription( int LtHandle,  
                           char *pString,  
                           int Len,  
                           double *pRefreshRate,  
                           int *pPixelRate,  
                           int *pHorizontalFreq,  
                           int *plsInterlace );
```

ARGUMENTS

LtHandle(in)

Handle to LT-XXX obtained from LT_OpenDevice ()

pString (out)

A string retrieving a short (32 characters) or long (128 characters) description of the detected input video signal. The length of the string is determined by *Len*

Len (in)

Length of pString. 0: short (32 characters), 1: long (128 characters)

pRefreshRate (out)

Video Refresh rate

pPixelRate (out)

Video pixel rate

pHorizontalFreq (out)

Horizontal frequency

plsInterlace (out)

Tells us whether input video is interlaced or not: 1: is interlaced, 0: is not interlaced

RETURN VALUE

Returns zero on success. A return value < 0 indicates an error.

COMMENTS

This function is more for debug purposes to check whether input signal is correct. The function can only be called after LT_UploadDeviceParameters() has been issued.

Unwanted parameters can be set to NULL.

USAGE EXAMPLE

```
char    VideoShortDescription[32];
char    VideoLongDescription[128];
double  RefreshRate;
int     PixelRate;
int     HorizontalFreq;
int     Status;
int     IsInterlace;

Status LT_UploadDeviceParameters(LtHandle);
else if( Status < 0 )
{
    printf( "LT_UploadDeviceParameters error : %d\n",Status );
    return Status;
}

LT_GetVideoDescription( LtHandle, VideoShortDescription, 0, &RefreshRate,
                        &PixelRate, &HorizontalFreq, &IsInterlace);

printf("VideoShortDescription: %s \n", VideoShortDescription);

LT_GetVideoDescription( LtHandle, VideoLongtDescription, 1, &RefreshRate,
                        &PixelRate, &HorizontalFreq, &IsInterlace);

printf("VideoLongtDescription: %s \n", VideoLongtDescription);
```

Text overlay

LT_SETOSD ()

This function sets an ASCII character based On-Screen-Display (OSD) or on screen overlay text.

```
int LT_SetOsd(int LtHandle,  
              int Index,  
              int X,  
              int Y,  
              char * String);
```

ARGUMENTS

LtHandle(in)
Handle to LT-XXX obtained from LT_OpenDevice()

Index (in)
Index of the word (0 to 3)

X (in)
Word x-pixel-position

Y (in)
Word y-pixel-position

String (in)
OSD character string

RETURN VALUE

Returns zero on success. A return value < 0 indicates an error.

COMMENTS

LT-XXX can set up to 4 words of 16 characters each. Each word can be placed at user defined position on top of the frame. It is important to note that the overlay is performed prior to compression and is thus encoded into the encoded video bitstream.

If the input video width is greater than 720, large characters (16x32) will be used for OSD, otherwise 8x16 characters are used.

This is a simple OSD implementation that only modifies the luma pixel values.

USAGE EXAMPLE

```
Char string[16] = {'T', 'h', 'i', 's', ' ', 'i', 's', ' ', 'a', ' ', 't', 'e', 's', 't', ' ', ' '};  
LT_SetOsd( LtHandle, 0, 32, 32, string );  
LT_SetOsd( LtHandle, 1, 32, 48, string );  
LT_SetOsd( LtHandle, 2, 32, 64, string );  
LT_SetOsd( LtHandle, 3, 32, 80, string );
```

Control functions

LT_STARTENC()

Starts the acquisition and compression of VC-1/H.264 video elementary stream video data.

```
int LT_StartEnc( int LtHandle );
```

ARGUMENTS

LtHandle(in)

Handle to LT-XXX obtained from LT_OpenDevice()

RETURN VALUE

Returns zero on success. A return value < 0 indicates an error.

COMMENTS

This function starts video acquisition. Inserting large execution delays between the LT_StartEnc() call and either LT_GetEncFrame() or LT_GetEncData() are not recommended as this could lead to excessive acquisition latency or internal buffer overflow.

USAGE EXAMPLE

```
if( LT_StartEnc( LtHandle ) )  
    return -1;  
  
while( Running && !Err )  
{  
    Err =LT_GetEncFrame(  
        LtHandle ,  
        pVBuf,  
        VBufSize,  
        &BytesTransferred,  
        &FrameType );  
}  
  
if( LT_StopEnc( LtHandle, 1 ) )  
    return -1;
```

LT_STOPENC()

Stops video acquisition at the next frame boundary.

```
int LT_StopEnc(    int LtHandle,  
                  int Flush );
```

ARGUMENTS

LtHandle(in)

Handle to LT-XXX obtained from LT_OpenDevice ()

Flush (in)

When non zero this command will stop the VC-1/H.264 stream and flush the remaining compressed video in the pipeline. If zero the user is responsible for flushing the remaining data.

RETURN VALUE

Returns zero on success. A return value < 0 indicates an error.

COMMENTS

As this function terminates acquisition at the end of the next frame you are assured of a clean stop.

After calling LT_StopEnc() with the flush argument zero, any remaining video elementary stream left in the processing pipeline should be read out using either LT_GetEncFrame() or LT_GetEncData().

When LT_StopEnc() is called with flush set there could be up to a one frame delay before the function returns.

USAGE EXAMPLE

```
if( LT_StopEnc( LtHandle, 0 ) )  
    return -1;
```


LT_INSERTIFRAME()

Next frame will be a Key Frame.

```
int LT_InsertIframe( int LtHandle );
```

ARGUMENTS

LtHandle(in)

Handle to LT-XXX obtained from LT_OpenDevice ()

RETURN VALUE

Returns zero on success. A return value < 0 indicates an error.

COMMENTS

During a record you may want to close current file and create a new one seamlessly. You need a Key Frame (also sometime referred to as Intra-Frame) to begin this new file. Use this function to force next frame to be a Key Frame.

This function can also be used to recover after a stream error knowing that key frames do not need any backward reference frame in time in order to be decoded.

There can be a delay of one or two frames according to the target bitrate and the time when function call occur.

USAGE EXAMPLE

```
if( LT_InsertIframe( LtHandle ) )  
    return -1;
```

LT_INSERTSKIPFRAME ()

Next frame will be a Skipped (dropped) Frame. This is only valid for VC-1 compression.

```
int LT_InsertSkipFrame (int LtHandle);
```

ARGUMENTS

LtHandle(in)

Handle to LT-XXX obtained from LT_OpenDevice ()

RETURN VALUE

Returns zero on success. A return value < 0 indicates an error.

COMMENTS

There can be a delay of one or two frames according to the target bitrate and the time when function call occur.

USAGE EXAMPLE

```
if( LT_InsertSkipFrame(LtHandle))  
    return -1;
```


LT_FORCESKIPFRAMES ()

All frames will be Skipped (dropped) from the moment this function is called until the function de-asserts frame skipping. This is only valid for VC-1 compression.

`int LT_ForceSkipFrames (int LtHandle, int On);`

ARGUMENTS

LtHandle(in)
Handle to LT-XXX obtained from LT_OpenDevice ()

On(in)
On/Off flag

RETURN VALUE

Returns zero on success. A return value < 0 indicates an error.

COMMENTS

This function can be used e.g. in case there is no enough bandwidth for streaming in order to relax the network.

There can be a delay of one or two frames according to the target bitrate and the time when function call occur.

USAGE EXAMPLE

```
LT_ForceSkipFrames (LtHandle, 1);  
...after a certain time  
LT_ForceSkipFrames (LtHandle, 0);
```

LT_STARTRAWCAPTURE()

Starts the acquisition of uncompressed (also referred to as raw) video images.

```
int LT_StartRawCapture(int LtHandle, int CaptureMode);
```

ARGUMENTS

LtHandle(in)

Handle to LT-XXX obtained from LT_OpenDevice ()

CaptureMode (in)

When this value is non-zero, hardware double buffering is enabled. Set to zero for single buffering. See comments for details.

RETURN VALUE

Returns zero on success. A return value < 0 indicates an error.

COMMENTS

This function starts uncompressed/raw acquisition.

Double buffering allows for higher capture rates as one image is captured while the other is being transferred to system memory. The disadvantage of the double buffer mode is that if the transfer rate is low, due to poor PCI or USB performance, the second buffer could contain an image acquired just after the first yet would only be transferred much later. This could result in images poorly spaced temporally.

Single buffering means that the image is only acquired when requested. As the second buffer is not present there can never be an old image. This mode is ideal for occasional snapshots or slow transfer busses. Without double buffering transfer the overall acquisition frame rate is reduced.

USAGE EXAMPLE

```
unsigned char *iBuf;  
int vRez = 720;  
int hRez = 1280;  
int ByteTransferred;  
.
```

```
.  
.   
iBuf = (unsigned char *)malloc( hRez * vRez * 2);  
if( iBuf == NULL )  
    return -1 ;  
.   
.   
.   
if( LT_StartRawCapture( LtHandle, 0 ) )  
return -1;  
  
while( running )  
{  
LT_GetRawImage( LtHandle, iBuf, hRez * vRez * 2, &ByteTransferred);  
PutImageSomewhere( iBuf ); //Pseudo function  
}
```

LT_STOPRAWCAPTURE()

Stops the acquisition of raw/uncompressed video.

```
int LT_StopRawCapture(int LtHandle, int Flush);
```

ARGUMENTS

LtHandle(in)

Handle to LT-XXX obtained from LT_OpenDevice ()

Flush (in)

When non zero this command will stop the raw/uncompressed video capture system and flush the remaining video in the pipeline. If zero the user is responsible for flushing the remaining data.

RETURN VALUE

Returns zero on success. A return value < 0 indicates an error.

COMMENTS

If using the single buffer capture mode and the LT_GetRawImage() function exclusively for transfer, flushing is not required as every image acquired is read.

USAGE EXAMPLE

```
if( iBuf == NULL )
    return -1 ;
.
.
.
if( LT_StartRawCapture( LtHandle, 0 ) )
    return -1;

while( running )
{
    LT_GetRawImage( LtHandle, iBuf, hRez * vRez * 2, &ByteTransferred);
    PutImageSomewhere( iBuf ); //Pseudo function
}
LT_StopRawCapture( LtHandle, 0);
```

LT_STARTAUDIO()

Starts the audio capturing and initializes audio compression.

```
int LT_StartAudio( int LtHandle );
```

ARGUMENTS

LtHandle(in)

Handle to LT-XXX obtained from LT_OpenDevice()

RETURN VALUE

Returns zero on success. A return value < 0 indicates an error.

COMMENTS

This function should be only called once at the start of audio capturing.

USAGE EXAMPLE

```
LT_StartAudio( LtHandle );
```

LT_STOPAUDIO()

Stops the audio capturing and releases resources associated with audio capturing.

```
int LT_StopAudio(    int LtHandle,  
                    int Flush );
```

ARGUMENTS

LtHandle(in)

Handle to LT-XXX obtained from LT_OpenDevice ()

Flush (in)

Flushes internal audio pipelines so that next audio start will fetch current data.

RETURN VALUE

Returns zero on success. A return value < 0 indicates an error.

COMMENTS

This function will reset some internal static parameters and release allocated memory.

USAGE EXAMPLE

```
LT_StopAudio( LtHandle, 1 );
```

Data transfer functions

LT_GETENCDATA()

This function retrieves any compressed video data in the LT-XXX pipeline without regard to frame boundaries.

```
int LT_GetEncData(  int LtHandle,
                   unsigned char * pBuffer,
                   int FetchSize,
                   int * pByteTransferred );
```

ARGUMENTS

LtHandle(in)

Handle to LT-XXX obtained from LT_OpenDevice ()

pBuffer (out)

Pointer to the user supplied (and allocated) video receive buffer. The buffer must be large enough to hold the data.

FetchSize (in)

Maximum number of bytes of video data to fetch. The fetched data size can be smaller than the value provided here.

pByteTransferred (out)

Pointer to the actual number of bytes transferred.

RETURN VALUE

Returns zero on success. A return value < 0 indicates an error.

COMMENTS

This function immediately pulls any compressed video data held in the output buffers up to the amount specified in FetchSize. This is suited to low latency applications where speedy data access is required without regard to frame boundaries.

Be sure that your FetchSize is not larger than your allocated buffer size.

Never assume that number of bytes actually transferred is the number requested with FetchSize.

USAGE EXAMPLE

```
unsigned char *pVBuf
int BytesTransferred;
int FrameType;
int Err;

.
.
.

pVBuf = (unsigned char *)malloc( 64 * 1024);

.
.
.

if( LT_StartEnc( LtHandle ) )
return -1;

while( Running && !Err )
{
    Err =LT_GetEncData( LtHandle,pVBuf,2048,&BytesTransferred );
}
```

LT_GETENCFRAME()

Acquire one VC-1 or H.264 video elementary stream image.

```
int LT_GetEncFrame(int LtHandle, unsigned char * pBuffer, int BufferSize, int *  
pByteTransferred, int * pFrameType);
```

ARGUMENTS

LtHandle(in)

Handle to LT-XXX obtained from LT_OpenDevice ()

pBuffer(out)

Pointer to the user supplied receive buffer. The buffer must be large enough to hold a complete frame (see comments below).

BufferSize(in)

Allocated size of the receive buffer (pBuffer) in bytes.

pByteTransferred(out)

Pointer to a value that will receive the actual number of video bytes in pBuffer.

pFrameType(out)

Pointer to a value that will receive the frame type where the frame types are 0: I (Intra) or 1:P (Inter). This may be set to NULL if frame type is not needed.

RETURN VALUE

Returns zero on success. A return value < 0 indicates an error.

COMMENTS

This function returns once a complete compressed frame has been placed in the receive buffer, unless there has been an error. If the buffer is not large enough to hold the frame, an error is returned. The buffer size is determined empirically and is a function of frame rate and desired bitrate. A 2048x2048x2 byte buffer should be large enough for worst case bitrate/frame rate combinations.

USAGE EXAMPLE

```
unsigned char *pVBuf
int BytesTransferred;
int FrameType;
int Err;
.
.
.
pVBuf = (unsigned char *)malloc( 2*1024*1024);
.
.
.
if( LT_StartEnc( LtHandle ) )
return -1;

while( Running && !Err )
{
    Err =LT_GetEncFrame( LtHandle, pVBuf,
                        2*1024*1024, &BytesTransferred,
                        &FrameType );
}
```

LT_GETRAWDATA()

Retrieves a chunk of raw/uncompressed YUV 4:2:2 (FOURCC UYVY) data from the LT-XXX acquisition buffer.

```
int LT_GetRawData( int LtHandle,  
                  unsigned char * pBuffer,  
                  int FetchSize,  
                  int * pByteTransferred );
```

ARGUMENTS

LtHandle(in)

Handle to LT-XXX obtained from LT_OpenDevice ()

pBuffer(out)

Pointer to the user supplied raw/uncompressed receive buffer.

FetchSize(in)

Desired number of raw/uncompressed bytes to receive in buffer

pByteTransferred(out)

Pointer to an integer value that will receive the actual number of bytes transferred.

RETURN VALUE

Returns zero on success. A return value < 0 indicates an error.

COMMENTS

This reads out uncompressed data in the YUV 4:2:2 format (FOURCC UYVY) from the capture buffer.

The FetchSize argument must not be larger than the allocated buffer size of *pBuffer*.

The minimum FetchSize is 2048 bytes. A value less than this will result in zero bytes being transferred.

Never assume that number of bytes actually transferred is the number requested with FetchSize.

USAGE EXAMPLE

```
unsigned char *iBuf;
int BytesTransferred;

.
.
.

iBuf = (unsigned char *)malloc( 1024 * 1024 );
if( iBuf == NULL )
    return -1 ;

.
.
.

while( running )
{
    LT_GetRawData( LtHandle, iBuf, 1024 * 1024, &BytesTransferred);
    PutDataSomewhere( iBuf ); //Pseudo function
}
```

LT_GETRAWIMAGE()

Retrieves a complete raw/uncompressed YUV 4:2:2 (FOURCC UYVY) image from the LT-XXX acquisition buffer.

```
int LT_GetRawImage( int LtHandle,  
                   unsigned char * pBuffer,  
                   int BufferSize,  
                   int * pByteTransferred);
```

ARGUMENTS

LtHandle(in)

Handle to LT-XXX obtained from LT_OpenDevice ()

pBuffer (out)

Pointer to the user supplied raw/uncompressed receive buffer. See comments below for sizing.

BufferSize (in)

Allocated size of raw/uncompressed receive buffer. See comments below for sizing.

pByteTransferred (out)

Pointer to an integer value that will receive the actual number of bytes transferred.

RETURN VALUE

Returns zero on success. A return value < 0 indicates an error.

COMMENTS

This reads out a full uncompressed frame in YUV 4:2:2 (FOURCC UYVY) format from the capture buffer. Each pixel requires two bytes of storage.

The buffer size must be at minimum (horizontal pixels) x (vertical pixels) x 2 bytes large.

USAGE EXAMPLE

```
unsigned char *iBuf;
int vRez = 720 ;
int hRez = 1280 ;
int ByteTransferred;
.
.
.
iBuf = (unsigned char *)malloc( hRez * vRez * 2);
if( iBuf == NULL )
    return -1 ;
.
.
.

while( running )
{
    LT_GetRawImage( LtHandle, iBuf, hRez * vRez * 2, &ByteTransferred);
    PutImageSomewhere( iBuf ); //Pseudo function
}
```


LT_GETAUDIO()

Retrieves a chunk of Stereo 16bit raw PCM audio data at 48Khz from hardware internal buffer if AUD_PCM_16BIT_STEREO_48KHZ is set.

Retrieves a chunk of stereo 16bit compressed WMA8 audio data at sampled at 48Khz if AUD_WMA8_STEREO_48KHZ is set.

Retrieves a chunk of stereo 16bit compressed AAC audio data at sampled at 48Khz if AUD_AAC_STEREO_48KHZ is set.

```
int LT_GetAudio(    int LtHandle,
                   unsigned char * pBuffer,
                   int BufferSize,
                   int * pByteTransferred,
                   int * pNoSamples );
```

ARGUMENTS

LtHandle(in)

Handle to LT-XXX obtained from LT_OpenDevice()

pBuffer (out)

This is a user defined buffer that will contain the audio data. The buffer must be large enough to hold a complete audio chunk (see comments below).

BufferSize (in)

Allocated size of pBuffer in bytes. (see comments)

pByteTransferred (out)

Number of audio data received (in bytes).

pNoSamples (out)

Number of audio samples/channel received. This value can be used to compute audio time stamps.

RETURN VALUE

Returns zero on success. A return value < 0 indicates an error.

COMMENTS

The amount of audio data retrieved depends on the system latency. The maximum possible allocated audio buffer is 2048x2048x2. The minimum buffer size is 2048bytes.

Before one can retrieve PCM audio, following parameter must be set:

```
LT_SetParam(LtHandle,LT_PARAMS_AUDIO_CODEC,AUD_PCM_16BIT_STEREO_48KHZ);
```

If you are using WMA compression:

Raw WMA8 data chunks will be fetched that does not contain WMA meta-information (no WMA container is generated). The audio data fetched cannot be played as such unless one adds a container or embeds the audio blocks in Directshow framework. Refer to our demo filter provided with LT-XXX SDK package or to our ASF dump example in the SDK.

The amount of audio data retrieved depends on the system latency. The maximum possible allocated audio buffer is 2048x2048x2.

Before one can retrieve WMA8 audio, following parameter must be set:

```
LT_SetParam( LtHandle, LT_PARAMS_AUDIO_CODEC, AUD_WMA8_STEREO_48KHZ );
```

If WMA8 is used with video in ASF container, the ASF packet size must be chosen as follows:

$\text{Asf_packet_size} = \text{audio_bitrate} / 187.5 + 28$

This restriction should be removed in later SDK releases.

USAGE EXAMPLE

```
unsigned char *pBuf;
int ByteTransferred;
int noSamples; //number of samples per channel grabbed

LT_SetParam( LtHandle, LT_PARAMS_AUDIO_CODEC, AUD_PCM_16BIT_STEREO_48KHZ
);
.
.
.
pBuf = (unsigned char *)malloc( 2048 * 2048 * 2);
if( iBuf == NULL )
    return -1 ;
.
.
.
LT_StartAudio( LtHandle );
while( running )
{
    LT_GetAudio( LtHandle, pBuf, 2048 * 2048 * 2, &ByteTransferred, &noSamples);
}
```

ASF/MPEG-TS Multiplexer functions

LT_INITTSMUXER ()

This function initializes MPEG-TS multiplexing functionality. MPEG-TS cannot be multiplexed with PCM or WMA audio.

```
int LT_InitTsMuxer ( int LtHandle);
```

ARGUMENTS

LtHandle(in)
Handle to LT-XXX obtained from LT_OpenDevice()

RETURN VALUE

Returns zero on success. A return value < 0 indicates an error.

COMMENTS

Initialization is needed for the creation of proper MPEG-TS streams.

USAGE EXAMPLE

```
LT_InitTsMuxer(LtHandle);
```

LT_INITASFMUXER ()

This function initializes ASF multiplexing functionality. ASF cannot be multiplexed with AAC audio.

```
int LT_InitAsfMuxer ( int LtHandle);
```

ARGUMENTS

LtHandle(in)

Handle to LT-XXX obtained from LT_OpenDevice()

RETURN VALUE

Returns zero on success. A return value < 0 indicates an error.

COMMENTS

Initialization is needed for the creation of proper ASF streams.

USAGE EXAMPLE

```
LT_InitAsfMuxer(LtHandle);
```

LT_GETMUXEDFRAME()

Fetch a compressed video elementary stream and optionally an audio chunk (raw or compressed) and multiplex audio and video in ASF or MPEG-TS file format.

```
int LT_GetMuxedFrame (    int LtHandle,
                          unsigned char * pBuffer,
                          int BufferSize,
                          int * pByteTransferred,
                          int * pFrameType );
```

ARGUMENTS

LtHandle(in)

Handle to LT-XXX obtained from LT_OpenDevice()

pBuffer(out)

Pointer to the user supplied video receive buffer. The buffer must be large enough to hold a complete frame (see comments below).

BufferSize(in)

Allocated size of the video receive buffer (pBuffer) in bytes.

pByteTransferred(out)

Pointer to a value that will receive the actual number of video bytes in *pBuffer*.

pFrameType(out)

Pointer to a value that will receive the frame type where the frame types are 0: I or 1: P. This may be set to NULL if frame type is not needed.

RETURN VALUE

Returns zero on success. A return value < 0 indicates an error.

COMMENTS

This function returns once a complete compressed frame has been placed in the receive buffer, unless there has been an error. If the buffer is not large enough to hold the frame, an

error is returned. The buffer size is determined empirically and is a function of frame rate and desired bitrate. An 8MByte byte buffer should be large enough for worst case bitrate/frame rate combinations.

The internal loop within the “encirslt.dll” API that runs LT_GetMuxedFrame() is as follows (pseudo code):

```
LT_GetEncFrame()  
LT_GetAudioPcm() or LT_GetAudioWma8()  
multiplex_audio_video_frame()
```

USAGE EXAMPLE

```
#define MAX_FR_SZ (2048*2048*2)  
  
int    wma_block_align;  
double aud_bitrate;  
int LtHandle    = 0;  
unsigned char *pAsfData = (unsigned char *)malloc(MAX_FR_SZ);  
int asfBytesXfered    = 0;  
int FrameType         = 0;  
int Status             = 0;  
int Vc1Cnt             = 0;  
.  
.  
.  
LT_SetParam(LtHandle,LT_PARAMS_VIDEO_INPUT,VID_COMPOSITE);  
  
// Input Resolution  
LT_SetParam(LtHandle,LT_PARAMS_VIDEO_RESOLUTION,AUTO);  
  
// Input Colorspace  
LT_SetParam(LtHandle,LT_PARAMS_VIDEO_COLORSPACE,VID_RGB);  
  
aud_bitrate = 192000;//in bits/sec  
LT_SetParam(LtHandle,LT_PARAMS_AUDIO_CODEC,AUD_WMA8_STEREO_48KHZ);  
LT_SetParam(LtHandle,LT_PARAMS_AUDIO_BITRATE,aud_bitrate);  
  
//asf config  
wma_block_align = (int)(aud_bitrate/187.5);  
LT_SetParam(LtHandle,LT_PARAMS_ASF_ENABLE_ASF,1);  
LT_SetParam(LtHandle,LT_PARAMS_ASF_PACKET_SIZE,wma_block_align +  
28);//wma_block_align + 28  
LT_SetParam(LtHandle,LT_PARAMS_ASF_PREROLL,33);  
.  
LT_InitAsfMuxer(LtHandle);  
.  
LT_StartEnc(LtHandle);  
LT_StartAudio(LtHandle);  
  
while( Vc1Cnt < 30 )  
{  
    int LtStatus = 0;  
    int junk;  
  
    printf("Vc1Cnt: %d\r", Vc1Cnt);
```

```
Status = LT_GetMuxedFrame(LtHandle, pAsfData, MAX_FR_SZ, &asfBytesXfered,
&FrameType);
// is an error occurred ?
if( Status < 0 )
{
    printf("\n");
    printf("ASF fetch error : %d\n",Status);
    return Status;
}

WriteDataSomewhere(asfBytesXfered,pFileVc1);//pseudo code
Vc1Cnt++;
}
```


LT_MUXFRAME()

This function only work together with VC-1 codec.

Take VC-1 and audio buffer and multiplex them in ASF file format using Enciris proprietary ASF multiplexer.

```
int LT_MuxFrame( int      LtHandle,
                 unsigned char * pVc1Buffer,
                 int      Vc1BuffSz,
                 int      FrameType,
                 unsigned char * pAudBuffer,
                 int      AudBuffSz,
                 unsigned char * pAsfBuffer,
                 int      MaxAsfBuffSz,
                 int      * pAsfByteTransferred);
```

ARGUMENTS

LtHandle(in)

Handle to LT-XXX obtained from LT_OpenDevice ()

pVc1Buffer (in)

This is a user defined buffer that will contain the VC-1 data. The buffer must be large enough to hold a complete audio chunk (see comments below).

Vc1BuffSz (in)

Allocated size of pVc1Buffer in bytes.

FrameType (in)

Frame type. 0: I-frame, 1: P-frame. This will be the frame type gathered from LT_GetEncFrame()

pAudBuffer (in)

This is a user defined buffer that will contain the audio data. The buffer must be large enough to hold a complete audio chunk (see comments below).

AudBuffSz (in)

Allocated size of pAudBuffer in bytes.

pAsfBuffer (out)

This is a user defined buffer that will contain the ASF data. The buffer must be large enough to hold a complete audio chunk (see comments below).

MaxAsfBuffSz (in)

Allocated size of pAsfBuffer in bytes.

pAsfByteTransferred (out)

Number of ASF bytes transferred.

RETURN VALUE

Returns zero on success. A return value < 0 indicates an error.

COMMENTS

To be on the safe side all buffers could be set to their theoretical maximum of 2048x2048x2 bytes.

Before using this function, user must implement the following loop:

LT_GetEncFrame()

LT_GetAudioPcm() or LT_GetAudioWma8()

LT_MuxFrame() can now be called.

USAGE EXAMPLE

```
#define MAX_FR_SZ 2048*2048*2
int wma_block_align;
double aud_bitrate;
int LtHandle = 0;
unsigned char *pVidData = (unsigned char *)malloc(MAX_FR_SZ);
unsigned char *pAudData = (unsigned char *)malloc(MAX_FR_SZ);
unsigned char *pAsfData = (unsigned char *)malloc(MAX_FR_SZ);
int asfBytesXfered = 0;
int FrameType = 0;
int Status = 0;
int Vc1Cnt = 0;
.
.
.

LT_SetParam(LtHandle,LT_PARAMS_VIDEO_INPUT,VID_COMPOSITE);

// Input Resolution
LT_SetParam(LtHandle,LT_PARAMS_VIDEO_RESOLUTION,AUTO);

// Input Colorspace
LT_SetParam(LtHandle,LT_PARAMS_VIDEO_COLORSPACE,VID_RGB);

aud_bitrate = 192000;//in bits/sec
LT_SetParam(LtHandle,LT_PARAMS_AUDIO_CODEC,AUD_WMA8_STEREO_48KHZ);
```

```

LT_SetParam(LtHandle,LT_PARAMS_AUDIO_BITRATE,aud_bitrate);

//asf config
wma_block_align = (int)(aud_bitrate/187.5);
LT_SetParam(LtHandle,LT_PARAMS_ASF_ENABLE_ASF,1);
LT_SetParam(LtHandle,LT_PARAMS_ASF_PACKET_SIZE,wma_block_align +
28);//wma_block_align + 28
LT_SetParam(LtHandle,LT_PARAMS_ASF_PREROLL,33);
.
LT_InitAsfMuxer(LtHandle);
.
LT_StartEnc(LtHandle);
LT_StartAudio(LtHandle);

while( Vc1Cnt < 30 )
{
    int LtStatus = 0;
    int junk;

    printf("Vc1Cnt: %d\r", Vc1Cnt);

    Status = LT_GetEncFrame(LtHandle, pVidData, MAX_FR_SZ, &vidBytesXfered,
&FrameType);
    // is an error occurred ?
    if( Status < 0 )
    {
        printf("Vc1 fetch error : %d\n",Status);
        return Status;
    }

    Status = LT_GetAudioPcm(LtHandle, pAudData, MAX_FR_SZ, &audBytesXfered);
    // is an error occurred ?
    if( Status < 0 )
    {
        printf("Aud fetch error : %d\n",Status);
        return Status;
    }

    Status = LT_MuxFrame(LtHandle,
                        pVidData,
                        vidBytesXfered,
                        FrameType,
                        pAudData,
                        audBytesXfered,
                        pAsfData,
                        MAX_FR_SZ,
                        &asfBytesXfered);

    if( Status < 0 )
    {
        printf("\n");
        printf("ASF fetch error : %d\n",Status);
        return Status;
    }

    WriteDataSomewhere(asfBytesXfered, pAsfData);//pseudo code
    Vc1Cnt++;
}

```

LT_GETASFINDEXBUFFER()

This is a helper function used if one needs to write ASF data to a file.

The function retrieves ASF index buffer (that has been generated during frame grabbing and multiplexing).

```
int LT_GetAsfIndexBuffer(    int LtHandle,
                           unsigned char * pBuffer,
                           int BufferSize,
                           int * pByteTransferred);
```

ARGUMENTS

LtHandle(in)

Handle to LT-XXX obtained from LT_OpenDevice ()

pBuffer(out)

This is a user supplied buffer that will contain the index data. 1024*1024 should be plenty for ours of recording.

BufferSize(in)

Allocated size of pBuffer. See comments for sizing.

pByteTransferred(out)

Actual length of index data contained in pBuffer

RETURN VALUE

Returns zero on success. A return value < 0 indicates an error.

COMMENTS

ASF index contains entry point information to the stream. Currently I-frame locations are stored each 32 frames (refer to ASF specification).

The index must be written at the end of the ASF file.

USAGE EXAMPLE

```
#define MAX_FR_SZ 2048*2048*2
#define ASF_PKT_SZ 2048
#define ASF_IDX_SZ 1024*1024

int    wma_block_align;
double aud_bitrate;
int LtHandle      = 0;
```

```

unsigned char *pAsfData = (unsigned char *)malloc(MAX_FR_SZ);
int asfBytesXfered = 0;
int FrameType = 0;
int Status = 0;
int Vc1Cnt = 0;
unsigned char *pIdxBuffer = (unsigned char *)malloc(ASF_IDX_SZ);
int IdxBufferSize = ASF_IDX_SZ;
int IdxByteTransferred;
unsigned char *pHdrBuffer = (unsigned char *)malloc(ASF_PKT_SZ);
int HdrBufferSize = ASF_PKT_SZ;
unsigned long long fileSize;
int HdrByteTransferred;
FILE *pFileVc1 = fopen ( "output.asf", "wb" );
.
.
.

LT_SetParam(LtHandle,LT_PARAMS_VIDEO_INPUT,VID_COMPOSITE);

// Input Resolution
LT_SetParam(LtHandle,LT_PARAMS_VIDEO_RESOLUTION,AUTO);

// Input Colorspace
LT_SetParam(LtHandle,LT_PARAMS_VIDEO_COLORSPACE,VID_RGB);

aud_bitrate = 192000;//in bits/sec
LT_SetParam(LtHandle,LT_PARAMS_AUDIO_CODEC,AUD_WMA8_STEREO_48KHZ);
LT_SetParam(LtHandle,LT_PARAMS_AUDIO_BITRATE,aud_bitrate);

//asf config
wma_block_align = (int)(aud_bitrate/187.5);
LT_SetParam(LtHandle,LT_PARAMS_ASF_ENABLE_ASF,1);
LT_SetParam(LtHandle,LT_PARAMS_ASF_PACKET_SIZE,wma_block_align +
28);//wma_block_align + 28
LT_SetParam(LtHandle,LT_PARAMS_ASF_PREROLL,33);
.
LT_InitAsfMuxer(LtHandle);
.
LT_StartEnc(LtHandle);
LT_StartAudio(LtHandle);

while( Vc1Cnt < 30 )
{
    int LtStatus = 0;
    int junk;

    printf("Vc1Cnt: %d\r", Vc1Cnt);

    Status = LT_GetMuxedFrame(LtHandle, pAsfData, MAX_FR_SZ, &asfBytesXfered,
    &FrameType);
    // is an error occurred ?
    if( Status < 0 )
    {
        printf("\n");
        printf("ASF fetch error : %d\n",Status);
        return Status;
    }

    WriteDataSomewhere(asfBytesXfered,pFileVc1);//pseudo code
    Vc1Cnt++;
}

```

```

}

//do ASF file postprocessing now

//this function retrieves the ASF index buffer
LT_GetAsfIndexBuffer(LtHandle, pIdxBuffer, IdxBufferSize, &IdxByteTransferred);

//append index to file
fwrite(pIdxBuffer,1,IdxByteTransferred,pFileVc1);

//get file size
fileSize = ftell(pFileVc1);

//this function retrieves the ASF top level header
LT_GetAsfHeaderObject(LtHandle, pHdrBuffer, HdrBufferSize, fileSize,
&HdrByteTransferred);

//rewind file pointer
rewind(pFileVc1);

//overwrite top level header object to .ASF file
fwrite(pHdrBuffer,1,HdrByteTransferred,pFileVc1);

LT_StopEnc(LtHandle,0);
LT_StopAudio(LtHandle,0);

```

LT_GETASFHEADEROBJECT()

This is a helper function used if one needs to write ASF data to a file. The function retrieves updated and internally built ASF “header object”. This header object must replace the placeholder-one generated at the beginning of recording.

```
int LT_GetAsfHeaderObject( int LtHandle,  
                           unsigned char * pBuffer,  
                           int BufferSize,  
                           unsigned long long fileSize,  
                           int * pByteTransferred );
```

ARGUMENTS

LtHandle(in)

Handle to LT-XXX obtained from LT_OpenDevice ()

pBuffer (out)

This is a user supplied buffer that will contain the ASF header object.

BufferSize (in)

Allocated size of pBuffer. The amount of bytes allocated to pBuffer should be greater or equal the number of bytes of ASF packets (refer to LT_PARAMS_ASF_PACKET_SIZE) if padding feature is used. Otherwise 2048bytes should be enough.

fileSize (in)

Actual length of recorded ASF file

pByteTransferred (out)

Actual length of returned pBuffer

RETURN VALUE

Returns zero on success. A return value < 0 indicates an error.

COMMENTS

This feature will only be used in case of ASF recording.

While recording ASF file, a dummy header will be written (with broadcast flag set to 1. See ASF specification for broadcast flag definition) and dummy values will be set for other parameters. After a file has been recorded successfully, this preliminary header should be

overwritten with a new one containing actual file size, timing information and so on...
Setting this header will allow for proper indexing of the file.

USAGE EXAMPLE

Please refer to the example given in `LT_GetAsfIndexBuffer()` section.

Miscellaneous functions

LT_SETEDID ()

This is a helper function that writes Extended Display Identification Data (EDID) for DVI/HDMI video sources to its corresponding EEPROM flash memory.

```
int LT_SetEdid(      int LtHandle,
                    unsigned char * pBuffer,
                    int BufferSize);
```

ARGUMENTS

LtHandle(in)

Handle to LT-XXX obtained from LT_OpenDevice ()

pBuffer (in)

This is a user supplied buffer that will contain the EDID data.

BufferSize (in)

Edid buffer size. Should be set to 256.

RETURN VALUE

Returns zero on success. A return value < 0 indicates an error.

COMMENTS

The EDID is a data structure provided by a computer display or any hardware that acts like a display (in this case the LT-XXX) to describe its capabilities to a video source (graphics card, video camera, etc...). The EDID includes manufacturer name and serial number, product type, timings as well as image resolutions supported by the LT-XXX.

LT-XXX acts like an HDMI display. In case of HDMI displays, VESA EDID block 0 and at least one CEA extension block is required. VESA EDID Version 1, Revision 3 is used in our case (it is also referred as E-EDID -> enhanced EDID).

Note: HDMI monitors require version 3 of the CEA extension in their EDID data structure.

Enciris Technologies can provide examples showing how to upload or read an EDID to or from our hardware internal EDID EEPROM.

USAGE EXAMPLE

```
char DevicePath[256];
int DeviceIndex = 0;
int LtHandle = 0;
int Status = LT_SUCCESS;

// Default EDID
unsigned char EdidBuffer[256] =
{
    ...specify EDID buffer here
};

// Load lt dll
HMODULE m_LtDll;
printf("--> LoadLtDll\n");
m_LtDll = LoadLtDll("encirislt.dll");

DeviceIndex = 0;
Status = LT_GetDevicePath(DevicePath, DeviceIndex);
if( Status < 0)
{
    printf("Error getting LT device path\n");
    return Status;
}

// OpenDevice
printf("--> OpenDevice\n");
LtHandle = LT_OpenDevice(DevicePath);
if( LtHandle < 0 )
{
    printf("Can't open %s - Error : %d\n",DevicePath,LtHandle);
    getchar();
    return LtHandle;
}

// Set Edid
printf("--> LT_SetEdid\n");
Status = LT_SetEdid(LtHandle,EdidBuffer,sizeof(EdidBuffer));
if( Status < 0 )
{
    printf("Edid programming error\n");
    getchar();
    return Status;
}
printf("<-- LT_SetEdid\n\n");

// Close Lautrec handle
printf("--> CloseDevice\n");
LT_CloseDevice(LtHandle);
FreeLtDll(m_LtDll);
```


LT_GETEDID ()

This is a helper function that read Extended Display Identification Data (EDID) for DVI/HDMI video sources from its corresponding EEPROM flash memory.

```
int LT_GetEdid(      int LtHandle,
                    unsigned char * pBuffer,
                    int BufferSize);
```

ARGUMENTS

LtHandle(in)

Handle to LT-XXX obtained from LT_OpenDevice ()

pBuffer (out)

This is a user supplied buffer that will contain the EDID data.

BufferSize (in)

Edid buffer size. Should be set to 256.

RETURN VALUE

Returns zero on success. A return value < 0 indicates an error.

COMMENTS

Enciris Technologies can provide examples showing how to upload or read an EDID to or from our hardware internal EDID EEPROM.

USAGE EXAMPLE

```
char DevicePath[256];
int DeviceIndex = 0;
int LtHandle = 0;
int Status = LT_SUCCESS;

// Default EDID
unsigned char EdidBuffer[256] =
{
    ...specify EDID buffer here
};

// Load lt.dll
```

```

HMODULE      m_LtDll;
printf("--> LoadLtDll\n");
m_LtDll = LoadLtDll("encirislt.dll");

DeviceIndex = 0;
Status = LT_GetDevicePath(DevicePath, DeviceIndex);
if( Status < 0)
{
    printf("Error getting LT device path\n");
    return Status;
}

// OpenDevice
printf("--> OpenDevice\n");
LtHandle = LT_OpenDevice(DevicePath);
if( LtHandle < 0 )
{
    printf("Can't open %s - Error : %d\n",DevicePath,LtHandle);
    getchar();
    return LtHandle;
}

// Set Edid
printf("--> LT_GetEdid\n");
Status = LT_GetEdid(LtHandle,EdidBuffer,sizeof(EdidBuffer));
if( Status < 0 )
{
    printf("Edid read error\n");
    getchar();
    return Status;
}
printf("<-- LT_GetEdid\n\n");

// Close Lautrec handle
printf("--> CloseDevice\n");
LT_CloseDevice(LtHandle);
FreeLtDll(m_LtDll);

```

LT_GETERROR ()

This is a helper function that has been designed to print to standard desktop a short description text corresponding to the error. This will however only work in DOS-Mode.

```
int LT_GetError( int Status);
```

ARGUMENTS

Status(in)

Error code. (please also refer to “LT-XXX API ERROR CODES” chapter)

RETURN VALUE

Returns zero on success. A return value < 0 indicates an error.

COMMENTS

Enciris Technologies can provide examples showing how to adapt this mechanism to a use case where no standard output is available.

USAGE EXAMPLE

```
char dp[256];
int Status = 0;

Status = LT_GetDevicePath( dp,0 ) ;
if( Status < 0 )
{
    LT_GetError(Status);
    return Status;
}
```

LT_GETVIDEOINPUTSTATUS ()

This is a helper function that returns a detailed status of the connected video sources. The function implements a mechanism that can detect framerate or resolution change issues.

```
int LT_GetVideoInputStatus ( int LtHandle, int VideoInputType, int * pVideoInputStatus);
```

ARGUMENTS

LtHandle (in)

Handle to LT-XXX obtained from LT_OpenDevice ()

VideoInputType (in)

Video input type. Please refer to comments for more details

pVideoInputStatus (out)

This is a 5 bit mask where each bit represents a particular video status. Please refer to the comments below for more details

RETURN VALUE

Returns zero on success. A return value < 0 indicates an error.

COMMENTS

This function can also be used in a separate thread. Don't call it too frequently when used in the same thread than the LT_StartRawCapture() or LT_StartEncCapture().

VideoInputType can be any of the following values:

VID_BOOTSCREEN¹

VID_SDI

VID_SDI1 (same than VID_SDI)

VID_SDI2²

VID_HD_DIGITAL

VID_HD_DIGITAL1(same as VID_HD_DIGITAL)

VID_HD_DIGITAL2³

VID_HD_ANALOG

VID_HD_ANALOG1(same as VID_HD_ANALOG)

VID_HD_ANALOG2⁴

VID_COMPOSITE

VID_COMPOSITE1(same as VID_COMPOSITE)

VID_COMPOSITE2⁵

VID_SVIDEO

Where:

¹Only valid for custom board

²Only valid for LT-122

³Only valid for LT-124 and LT-125

⁴ Only valid for custom board

⁵ Only valid for custom board

The returned "pVideoInputStatus" 5 bit mask is decoded as follows:

```
#define VIDSTATUS_NULL (0)
#define VIDSTATUS_INPUT_AVAILABLE (1 << 0)
#define VIDSTATUS_INPUT_ACTIVE (1 << 1)
#define VIDSTATUS_SIGNAL_PRESENT (1 << 2)
#define VIDSTATUS_SIGNAL_STABLE (1 << 3)
#define VIDSTATUS_SIGNAL_CHANGED (1 << 4)
```

Where:

VIDSTATUS_INPUT_AVAILABLE :

Video input exists and is available (Example : If DVI-D active in others threads/channels DVI-A become unavailable)

VIDSTATUS_INPUT_ACTIVE :

The video input is currently used in other thread/channel.

VIDSTATUS_SIGNAL_PRESENT :

Video is present on selected input.

VIDSTATUS_SIGNAL_STABLE :

Video is stable on selected input.

VIDSTATUS_SIGNAL_CHANGED :

Video resolution change between now and the last video input configuration.

USAGE EXAMPLE

```
#include <stdio.h>
#include <stdlib.h>
#include <direct.h>
#include "enciris_ltlb.h"
#include "enciris_lt.h"

int main(int argc, char* argv[])
{
    // Device handling
    char DevicePath[256];
    int DeviceIndex = 0;
    int LtHandle = 0;
    int Status = LT_SUCCESS;
    int i = 0;

    // Load lt dll
    HMODULE m_LtDll;
    printf("--> LoadLtDll\n");
```

```

m_LtDll = LoadLtDll("encirislt.dll");
if( !m_LtDll ){printf("Library encirislt.dll not found !\n"); return -1;}

LT_Log( LOG_LEVEL_NULL, NULL );

////////////////////////////////////
// Get path to LT

DeviceIndex = 0;
Status = LT_GetDevicePath(DevicePath, DeviceIndex);
if( Status < 0 )
{
    printf("Error getting LT device path\n");
    LT_GetError(Status);
    return Status;
}

////////////////////////////////////
// OpenDevice

printf("--> OpenDevice\n");
LtHandle = LT_OpenDevice(DevicePath);
if( LtHandle < 0 )
{
    printf("Can't open %s - Error : %d\n",DevicePath,LtHandle);
    LT_GetError(LtHandle);
    return LtHandle;
}
printf("<-- OpenDevice\n\n");

while( 1 )
{
    int VideoInput = 0;
    int VideoStatus = 0;

    switch( i )
    {
        // Test every inputs
        default : i = 0;                printf("\r");
        case 0 : VideoInput = VID_SDI1;printf("SDI1(");break;
        case 1 : VideoInput = VID_SDI2;printf("SDI2(");break;
        case 2 : VideoInput = VID_HD_DIGITAL;printf("DVI-I(");break;
        case 3 : VideoInput = VID_HD_ANALOG;printf("DVI-A(");break;
        case 4 : VideoInput = VID_COMPOSITE1;printf("C1(");break;
        case 5 : VideoInput = VID_COMPOSITE2;printf("C2(");break;
        case 6 : VideoInput = VID_SVIDEO;printf("SV(");break;
    }

    Status = LT_GetVideoInputStatus(LtHandle, VideoInput, &VideoStatus);
    if( Status < 0 )
    {
        LT_GetError(Status);
    }

    if( VideoStatus & VIDSTATUS_INPUT_AVAILABLE )
    {
        printf("*");//Video input exists and is available
    }
    else
    {
        printf(" ");//Video input is not available
    }

    if( VideoStatus & VIDSTATUS_INPUT_ACTIVE )
    {
        printf(".");//The video input is currently used in other thread/channel
    }
}

```

```

        else
        {
            printf(" "); //The video input is inactive
        }

        if( VideoStatus & VIDSTATUS_SIGNAL_PRESENT )
        {
            printf("P"); //Video is present on selected input
        }
        else
        {
            printf(" "); //Video is not present on selected input
        }

        if( VideoStatus & VIDSTATUS_SIGNAL_CHANGED )
        {
            printf("C"); //Video resolution change between now and the last video input
configuration
        }
        else
        {
            printf(" "); //Video resolution has not change between now and the last
video input configuration.
        }

        printf(" ");

        i++;
    }

    printf("--> CloseDevice\n");
    LT_CloseDevice(LtHandle);
    FreeLtDll(m_LtDll);
    printf("<-- CloseDevice\n\n");

    printf("\nType enter to exit program\n");
    getchar();
    return 0;
}

```

LT_LOG ()

This is a helper function that logs error and status information either to standard output or to a file.

```
int LT_Log(int Mask, char *FilePath, void *Reserved);
```

ARGUMENTS

Mask(in)

Sets 16bit log location and log level mask. (LOG_TYPE_PRINTF, LOG_TYPE_FILE, LOG_LEVEL_NULL, LOG_LEVEL_INFO, LOG_LEVEL_DEBUG, LOG_LEVEL_ERROR and LOG_LEVEL_ALL) are valid values. One can “OR” between several values

FilePath (in)

Path to the file containing the log if file destination has been chosen

Reserved (in)

This is a reserved value and should always be set to NULL

RETURN VALUE

Returns zero on success. A return value < 0 indicates an error.

COMMENTS

It is recommended to use LOG_LEVEL_ALL in case you request tech support from Enciris Technologies in order to provide as much information as possible.

Note:

```
LT_Log( LOG_LEVEL_NULL, NULL , NULL); //disables all logs
```

```
LT_Log( LOG_TYPE_PRINTF | LOG_LEVEL_ALL, NULL , NULL); //enables all logs on standard output. This is the default configuration.
```

USAGE EXAMPLE

```
char log_FilePath[272];  
  
Status = LT_Log(LOG_TYPE_FILE | LOG_LEVEL_ALL, log_FilePath, NULL);  
if( Status < 0 )
```

```
{  
    LT_GetError(Status);  
    return Status;  
}
```

LT_INITIALIZEDEVICE ()

This is a helper function that forces the .bit FPGA firmware file uploading. The FPGA firmware (*.bit) also referred to as FPGA bitstream is only uploaded once after. As long as power is applied to the Enciris card, no bitstream is uploaded. This is done to save time for the device to get ready. LT_InitializeDevice() forces the upload of the FPGA bitstream. This could for example be useful in a context where the user wants to test a development bitstream provided by Enciris for debug purposes.

```
int LT_InitializeDevice (int LtHandle);
```

ARGUMENTS

LtHandle(in)

Handle to LT-XXX obtained from LT_OpenDevice ()

RETURN VALUE

Returns zero on success. A return value < 0 indicates an error.

COMMENTS

This function is generally used for debugging. There is not real reason to use it in a normal context.

USAGE EXAMPLE

```
Status = LT_InitializeDevice(LtHandle);
if( Status < 0)
{
    LT_CloseDevice(LtHandle);
    FreeLtDll(m_LtDll);
    return Status;
}
```

LT_GETSTILLIMAGE ()

This is a helper function that adds support for capturing of still images, say every 10 second, to be used for presentation of recorded timeline used e.g for presentation of recorded timeline. This function returns YUV 4:2:2 (FOURCC UYVY) uncompressed raw image of size width*height*2

```
int LT_GetStillImage (int LtHandle, unsigned char* pBuffer, int BufferSize, int *
pByteTransferred);
```

ARGUMENTS

LtHandle(in)

Handle to LT-XXX obtained from LT_OpenDevice ()

pBuffer (out)

user buffer that contains the still image captured

BufferSize (in)

User buffer size. Must be at least of size in_video_width * in_video_height * 2 to receive UYVY uncompressed raw image

pByteTransferred (out)

Actual size of returned buffer (in_video_width * in_video_height * 2). If this is less than in_video_width * in_video_height * 2, then an error occurred.

RETURN VALUE

Returns zero on success. A return value < 0 indicates an error.

COMMENTS

This function can be used in parallel with the streaming of compressed video and the raw uncompressed video without interference. It can be used in a separate thread.

USAGE EXAMPLE

```
//This example shows how to capture a still image and convert it to bitmap
using ffmpeg library programatically
int img_width = 1920;
int img_height = 1080;
unsigned char *pStillData = (unsigned char *)malloc(MAX_FR_SZ);
int vidBytesXfered = 0;
```

```

//prepare destination RGB24
int rgbBufferSize =img_width*img_height*3;
AVFrame *pFrameRGB24;
pFrameRGB24=avcodec_alloc_frame();
uint8_t *out_buffer;
out_buffer=new uint8_t[avpicture_get_size(PIX_FMT_RGB24,img_width,
img_height)];
avpicture_fill((AVPicture *)pFrameRGB24, out_buffer,
PIX_FMT_RGB24,img_width, img_height);

//prepare source UYVY
AVFrame *pFrameUYVY422;
pFrameUYVY422=avcodec_alloc_frame();
uint8_t *out_buffer1;
out_buffer1=new uint8_t[avpicture_get_size(AV_PIX_FMT_UYVY422,img_width,
img_height)];
avpicture_fill((AVPicture *)pFrameUYVY422, out_buffer1,
AV_PIX_FMT_UYVY422,img_width, img_height);

//point to the source video
LT_GetStillImage(LtHandle, pStillData , MAX_FR_SZ, &vidBytesXfered);
pFrameUYVY422->data[0] = pStillData;

//convert to RGB24bit
SwsContext * img_convert_ctx = sws_getContext(img_width, img_height,
AV_PIX_FMT_UYVY422,img_width, img_height, AV_PIX_FMT_BGR24, SWS_BICUBIC,
NULL, NULL, NULL);
sws_scale(img_convert_ctx, (const uint8_t* const*)pFrameUYVY422->data,
pFrameUYVY422->linesize, 0, img_height, pFrameRGB24->data, pFrameRGB24-
>linesize);

//Raw RGB/BGR data
//fwrite(pFrameRGB24->data[0],rgbBufferSize,1,outfile);

//convert to BMPBMP
AVCodec *ecodec = avcodec_find_encoder(AV_CODEC_ID_BMP);
//also works AVCodec *ecodec = avcodec_find_encoder_by_name ("bmp");
AVCodecContext *ecctx = avcodec_alloc_context3(ecodec);
if (!ecctx)
{
    printf("Failed to get encoder context\n");
    exit(1);
}
avcodec_get_context_defaults3(ecctx, NULL);
ecctx->width =img_width;
ecctx->height = img_height;
ecctx->pix_fmt = AV_PIX_FMT_BGR24;
if (avcodec_open2(ecctx, ecodec, NULL) < 0)
{
    printf("Failed to open encoder\n");
    exit(1);
}

AVPacket pkt;
av_init_packet(&pkt);
pkt.data = NULL;
pkt.size = 0;

```



```

int gotPacket;
avcodec_encode_video2 (ecctx,&pkt,pFrameRGB24, &gotPacket);
outfile = fopen("snapshot.bmp", "wb");//output snapshot
fwrite(pkt.data,pkt.size,1,outfile);
fclose(outfile);

av_free_packet (&pkt);

```

LT-XXX API ERROR CODES

Each API function returns an error code. If 0, no error occurred. If inferior to 0, an error occurred.

In order to be able to understand the error codes, following table is given (see enciris_lt.h):

ERROR Code	Value	Description
LT_SUCCESS	0	Success code. Everything went well
LT_DRV_ERROR_PD	-1004	Set DDR phase delay error
LT_DRV_ERROR_MCPSE T	-1008	Error setting MCP* Profile into driver
LT_DRV_ERROR_I2C	-1012	Error while writing to an I2c register
LT_DRV_ERROR_SPI	-1016	Error while writing firmware binary file: *.bit
LT_DRV_ERROR_CFG	-1020	Could not load LT-XXX configuration
LT_DRV_ERROR_CMD	-1024	Error loading MCP* command
LT_DRV_ERROR_RSTL	-1028	Reset Lautrec error. Error while resetting the board
LT_DRV_ERROR_RSTP	-1032	Reset peripheral. Error resetting peripheral chips
LT_BAD_POINTER	-1034	Error: pointer provided to LT_GetVideoInputStatus() function is NULL
LT_DRV_ERROR_FWCTR	-1036	Firmware control error at new SPI cycle
LT_DRV_ERROR_CSI	-1040	Error reading internal status register
LT_DRV_ERROR_OSD	-1044	Error loading On Screen Display (OSD) command
LT_DRV_ERROR_STAT	-1048	Error reading MCP* status
LT_DRV_ERROR_REG	-1052	Error writing/reading a register
LT_DRV_ERROR_SNROM	-1054	Error while trying to read board serial number
LT_DRV_ERROR_LEVEL	-1056	Error reading data fullness register

LT_DRV_ERROR_RATECOUNTER	-1058	Error reading framerate counter register
LT_DRV_ERROR_FWINFO	-1059	Error while trying to read firmware information
LT_DRV_ERROR_GETENC	-1060	Error fetching compressed video data
LT_DRV_ERROR_GETAUD	-1064	Error fetching audio data
LT_DRV_ERROR_GETRAW	-1068	Error fetching raw uncompressed data
LT_DRV_ERROR_GETSTILL	-1070	Error fetching still image
LT_DRV_ERROR_PROD_VER	-1076	Error getting USB product version from KMDF
LT_DRV_ERROR_FAN	-1078	Error switching fan on or off
LT_DRV_ERROR_HSBC	-1080	Error writing color enhancement parameters
LT_DRV_ERROR_CSC	-1081	Error while programming Color Space Conversion (CSC) parameter
LT_DRV_ERROR_BRC	-1084	Error while programming Bit Rate Control parameter
LT_DRV_ERROR_CONTEXTSET	-1086	Error while writing context values related to this compression channel
LT_DRV_ERROR_CONTEXTGET	-1088	Error while reading context values related to this compression channel
LT_DRV_EDID_SELECT	-1090	Error while selecting FPGA internal EDID
LT_DRV_ERROR_DEVICE_RESET	-1094	Error resetting device
LT_DRV_EDID_WR	-1092	Error while enabling/disabling EDID write
LT_BAD_DEV_INDEX	-1104	Bad device index (Device has not been seen. Please check if device is present)
LT_BAD_CHANNEL_INDEX	-1105	Channel index out of range
LT_BAD_FW_SELECTION	-1107	Firmware selection index unknown
LT_BAD_CHANNEL_TYPE	-1106	Channel type is out of allowed range
LT_BAD_DEV_PATH	-1108	Error using "CreateFile" command
LT_BAD_LOG_PATH	-1109	Logging file path is not valid
LT_DEVICE_ALREADY_OPENED	-1110	An instance of the device has already been opened
LT_CHANNEL_ALREADY_OPENED	-1111	The board/channel is already in use
LT_BAD_HANDLE	-1112	Bad device handle
LT_DEVICE_ALREADY_INUSE	-1113	Could not reinitialize/change firmware because the device is already in use

LT_BAD_PARAM_KEY	-1116	User entered a non-supported parameter key
LT_BAD_PARAM_VALUE	-1120	User entered a non-supported parameter value
LT_BAD_VIDEO_MODE	-1121	This video mode is not supported
LT_BAD_VIDEO_INPUT	-1122	This video input type is not supported
LT_BAD_AUDIO_MODE	-1126	This audio mode is not allowed
LT_BAD_FRAMERATE	-1128	This target frame rate is not allowed
LT_BAD_BUFFERSIZE	-1132	User provided buffer is invalid or too small. Please try to increase allocated buffer.
LT_BAD_FILE_PATH	-1135	BAD input bitstream FILE PATH
LT_BAD_FILE_TYPE	-1136	BAD FILE TYPE
LT_BAD_FILE_FORMAT	-1137	BAD FILE FORMAT
LT_ERROR_FILE_OPEN	-1140	ERROR OPENING File
LT_ERROR_FILE_READ	-1141	ERROR FILE READ
LT_ERROR_I2C_READ	-1150	ERROR I2C READ
LT_ERROR_I2C_WRITE	-1151	ERROR I2C WRITE
LT_FW_ERROR_FILEOPEN	-1200	Firmware file could not be opened
LT_FW_ERROR_FILEREAD	-1204	Firmware file could not be read from hard drive. Check if the *.bit firmware file exist and is not in use.
LT_FW_ERROR_SIZE	-1208	Firmware file was not written entirely
LT_FW_ERROR_FILEBAD	-1212	Ftell returned negative file length. The file may be in use.
LT_FW_ERROR_LOAD	-1216	After writing firmware a status register is checked to see whether firmware has been written properly. This check failed. Please try again to open the device.
LT_FW_ERROR_MEM	-1220	Firmware memory allocation failed
LT_CHANNEL_MUTEX_TIMEOUT	-1241	Mutex timeout error
LT_DDR_CALIB_ERROR	1250	DDR Controller calibration failed. This may lead to data corruption
LT_EDID_CHECKERROR	-1224	EDID mismatch. A comparison of the EDID programmed on EEPROM and the one provided by user failed.
LT_SDI_CHECKERROR	-1228	SDI external daughter board module not present or not plugged correctly

LT_CHANNEL_MUTEX_ERROR	-1240	Error while releasing inter process critical section. Error occurred while calling WaitForSingleObject() or CreateMutex()
LT_DDR_CALIB_ERROR	-1250	DDR memory calibration error
LT_ACQ_OVERFLOW	-2000	Acquisition buffer overflow
LT_ENC_EMPTY	-2002	There is no compressed video in internal buffers
LT_ENC_OVERFLOW	-2004	<p>Probably decoder CPU is not fast enough to decode the data and internal overflow occurred. Use lower target bitrate when using bitrate control mode or higher quantization factor when using constant quality mode. Video hardware internal cache buffer overflow occurred!</p> <p>Several options could be helpful to avoid this issue in future:</p> <ol style="list-style-type: none"> 1- Use more powerful computer. 2- Lower the target bitrate. 3- Lower the video resolution using the downscaling features provided by the driver.
LT_RAW_EMPTY	-2006	Raw buffer is empty.
LT_AUD_EMPTY	-2008	Audio buffer is empty
LT_AUD_OVERFLOW	-2010	Audio buffer overflow
LT_STILL_EMPTY	-2013	No still image could be captured
LT_NO_VIDEO_SIGNAL	-3000	No video input signal has been detected. Check if LT-xxx is properly connected and driver has been installed successfully. Check basic functionality by using colorbar debug test mode in demo application.
LT_UNSUPPORTED_VIDEO_SIGNAL	-3004	The video signal is not supported. A video signal has been detected at input but this video mode is unknown. Please check debug output of our demo application that gives details of video signal.
LT_UNSTABLE_VIDEO_SIGNAL	-3006	A video signal is present but there are instabilities on it
LT_UNAVAILABLE_VIDEO_SIGNAL	-3007	Video signal exists but cannot currently be handled because of a conflict with another video signal
LT_VIDEOMODE_UNKNOWN	-3008	Bad parameter value of: LT_PARAMS_VIDEO_RESOLUTION key
LT_ERROR_FFMPEG_DLL	-3024	avcodec-52.dll or avutil-50.dll could not be found. If using WMA8 audio encoding you need to put these dll's

		in the executables directory. LT-XXX Installation program will put them in system directory.
LT_ERROR_AAC_SET_CONFIG	-3016	AAC Configuration has returned an error
LT_ERROR_AAC_ENCODING	-3020	Error while encoding audio chunk. AAC encoder has returned an error
LT_ERROR_USB_FIRMWARE	-3028	Error while trying to update Cypress USB firmware
LT_ERROR_WMA_ENCODING	-3022	Error while encoding audio chunk
LT_ERROR_USB_GPIO	-3032	Error while reading/writing Cypress chip internal USB register
LT_ERROR_NO_MUXER_SELECTED	-3036	you must call either LT_SetParam(LtHandle,LT_PARAMS_TS_ENABLE_TS,1) or LT_SetParam(LtHandle,LT_PARAMS_ASF_ENABLE_ASF,1)

*MCP: Master Command Processor. This is an internal state machine that possesses some macro commands like “program i2c register”, “set FAN on” etc...

CONCLUSION

For more information, please also refer to our numerous examples provided in the LT-XXX Software Development Kit (SDK) and the directshow documentation.

Note that Enciris Technologies will be pleased to provide further examples showing special use cases of our API not covered in the SDK.

ENCIRIS TECHNOLOGIES

22 Avenue de l'Europe
81600 Gaillac
France

Tel : +33(0)5 81 18 01 12
info@enciris.com
<http://www.enciris.com>

Copyright © 2016 Enciris Technologies, SAS.

Other companies' product names that may be used in this publication are for identification purposes only and may be trademarks of their respective companies.