LTT2API - Reference

Version 3.2.xx

LTT GmbH
Friedrich-Bergius-Ring 15
D-97076 Würzburg
www.tasler.de

Tel. +49-931-359 61-0
Fax +49-931-359 61-50

This handbook has been created to the best of our knowledge. The information contained herein is given without guarantee and may be changed without notice. The software described is covered by the included software licence.

Friedrich-Bergius-Ring 15
D-97076 Würzburg
Germany
E-Mail: info@tasler.de
Internet: www.tasler.de

Content

Introduction

To integrate LTT measurement systems with third party applications the LTT2API-DLL was developed. Using the DLL opens a wide range of possibilities for efficient use of the LTT-Hardware.
This chapter describes the programming interface **LTT2API Version 3.2.x**.

Every function is described using the following format:

<Function Name>

[Device]

Declaration:

```
<Declaration of the function>
```

Description:
<Description of the function and explanation of the use of the function>

Input:
<specifies and describes the function parameters>

Return:
<specifies the return value>

Note: The note [Device] in the 2nd line after the <Function Name> indicates a direct communication with the LTT-System.

Chapter 1:
Fundamental Concepts

The API is designed to manage one or a group of LTT devices in an efficient manner. The most important design goals are stability and performance. The calls are described in STDCALL calling convention.

The use of the API is divided in several steps:

Initialization:

The call **LTTInit** must be the first call to the API. With this call the internal resources are initialised and the system is scanned for connected LTT devices. Note: This function should be used ***only once*** at the beginning.

Setup:

The devices can now be configured with a certain setup, as turn on several channels, select its input type and range or select a trigger. Also the amount of data has to be specified. These calls don't communicate directly with the devices. Instead, all setup calls build up a certain configuration which is passed to the devices with **LTTUploadParameters**. This call does an integrity check to see if the configuration is a valid one.

Transfer:

The devices will be started with **LTTStart**, which basically means that the trigger will be active or in the case of no trigger, the measurement process of the devices begins.
Now the transfer of the data from the devices to the RAM-buffer/File can be done in two ways. In both cases the transfer is invoked with **LTTTransfer2Buffers**.

BLOCKED_MODE:
The call LTTTransfer2Buffers returns when the transfer of the data to the RAM-buffer/File is completed. This is the easiest way, and the most used one.

NONE_BLOCKED_MODE:
The call **LTTTransfer2Buffers** returns **immediately** after invoking the transfer. Now the transfer to the RAM-buffer/File needs the cooperation of the application. The transfer must be monitored with **LTTTransferStatus** to catch when the transfer is completed or interrupted.

In STREAM-mode the internally used buffer is mostly smaller then the requested transfer amount of data, so this buffer is used like a ring-buffer. To avoid the overwriting of already in the buffer transferred data, the application must read out blocks of already transferred data. If the call **LTTTransferStatus** signals that there is an amount of data ready for read-out, then the start index and the length of the block inside the buffer must be obtained with **LTTData_GetBlock**. When the

processing of the block is finished, **LTTData_BlockDone** must be called. Now go back to monitoring the transfer with **LTTTransferStatus**.

Deinitialization:

To close all internal resources, call **LTTDeInit**.

Other calls:

The API provides a number of different informative calls. They are not necessary for the data transfer, but quite a lot of information about the status and the devices can be obtained.

1.1 Source code samples

In the following a source code sample for each of the transfer modes is provided. Both sample are written in plain C:

1.1.1 BLOCKED_MODE:

```
#include <ltt2api.h>

uint32_t ch_an, ch_dig;
short p_data[128*1024];
int32_t dev_cnt, ret, seq;

...
dev_cnt = LTTInit( &ch_an, &ch_dig, NULL, NULL );

...
// channel 0 on, +/-5V range, single ended DC
LTTAnalogChannel( 0, 1, 5000, LTT_CHNL_24_COUP_VOLT_SE_DC_BNC, 0, 0 );
// channel 3 on, +/-500mV range, diff. ended DC
LTTAnalogChannel( 3, 1, 500, LTT_CHNL_24_COUP_VOLT_DE_DC_BNC, 0, 0 );

// trigger at ch 0, LEVEL, 2V-threshold,
// pos. slope, 200mV sensitivity
LTTTrigger( 1, 0, 0, 0, 0, 0.4, 0.04 );

// sample rate: 1 Mhz <==> 1000 ns
LTTSampleTime( 1000, 0 );

// RAMBuffer: 32kS per channel ==> 2 * 32 * 1024 * 2 = 128 kB
seq =  32 * 1024;
LTTRAMBufferSetting( &p_data, seq, 2 );

// submit to device(s)
if( LTTUploadParameters() == 0 ) {
      // start LTT device(s)
      LTTStart();

      // invoke transfer
      lAmount = LTTTransfer2Buffers( 0, LTT_TRANSFER_BLOCKED );
}

...
// clean up
LTTDeInit();
```

1.1.2 NON_BLOCKED_MODE:

```
#include <ltt2api.h>

uint32_t ch_an, ch_dig;
short p_data[128*1024];
int32_t dev_cnt, ret, seq, status;

...
dev_cnt = LTTInit( &ch_an, &ch_dig, NULL, NULL );

...
// channel 0 on, +/-5V range, single ended DC
LTTAnalogChannel( 0, 1, 5000, LTT_CHNL_24_COUP_VOLT_SE_DC_BNC, 0, 0 );
// channel 3 on, +/-500mV range, diff. ended DC
LTTAnalogChannel( 3, 1, 500, LTT_CHNL_24_COUP_VOLT_DE_DC_BNC, 0, 0 );

// trigger at ch 0, LEVEL, 2V-threshold,
// pos. slope, 200mV sensitivity
LTTTrigger( 1, 0, 0, 0, 0, 0.4, 0.04 );

// sample rate: 1 Mhz <==> 1000 ns
LTTSampleTime( 1000, 0 );

// RAMBuffer: 32kS per channel ==> 2 * 32 * 1024 * 2 = 128 kB
seq =  32 * 1024;
LTTRAMBufferSetting( &p_data, seq, 2 );

// submit to device(s)
if( LTTUploadParameters() == 0 ) {
      // start LTT device(s)
      LTTStart();

      // invoke transfer
      LTTTransfer2Buffers( 0, LTT_TRANSFER_NONEBLOCKED );

      // monitor transfer
      status  = 0;
      while( status <= 0 ) {
            status = LTTDataTransferStatus();

            if( status == 0 )
                  // nothing todo
                  continue;

            else if( status == -1 )
                  // waiting for trigger
                  DoSomethingWhileNotTriggered();

            else if( status < 0 )
                  // ERROR
                  break;
      }

      if( status > 0 ) {
            // do something with the data
      }
}

...
// clean up
LTTDeInit();
```

1.1.3 STREAM Transfer:

```
#include <ltt2api.h>

uint32_t ch_an, ch_dig;
short p_data[128*1024];
int32_t dev_cnt, ret, seq;

...
dev_cnt = LTTInit( &ch_an, &ch_dig, NULL, NULL );

      ...
// channel 0 on, +/-5V range, single ended DC
LTTAnalogChannel( 0, 1, 5000, LTT_CHNL_24_COUP_VOLT_SE_DC_BNC, 0, 0 );
// channel 3 on, +/-500mV range, diff. ended DC
LTTAnalogChannel( 3, 1, 500, LTT_CHNL_24_COUP_VOLT_DE_DC_BNC, 0, 0 );

// trigger at ch 0, LEVEL, 2V-threshold,
// pos. slope, 200mV sensitivity
LTTTrigger( 1, 0, 0, 0, 0, 0.4, 0.04 );

// sample rate: 1 Mhz <==> 1000 ns
LTTSampleTime( 1000, 0 );

// STREAM: 32kS per channel ==> 2 * 32 * 1024 * 2 = 128 kB
seq =  32 * 1024;
LTTSTREAMTransfer( 0, seq );

// submit to device(s)
if( LTTUploadParameters() == 0 ) {
      // start LTT device(s)
      LTTStart();

      // invoke transfer
      ret = LTTStream2Buffer( p_data, seq, LTT_TRANSFER_BLOCKED );
}

...
// clean up
LTTDeInit();
```

Chapter 2:
Function Reference

2.1 Initialisation

2.1.1 LTTInit

Changed

[Device]

Declaration:

```
int32_t __stdcall   LTTInit(
      int32_t *p_analog_cnt,
      int32_t *p_digital_cnt,
      int32_t *p_rpm_cnt,
      int32_t *p_digio_cnt
);
```

Description:

Initializes internal resources and searches for all connected LTT devices. Supported are the transient recorder LTT-184/186 as well as new sensorcorder LTT-180/182.
The parameters are optional and a NULL-Pointer can be passed also;
The devices found are always connected after initialisation.

Input:

p_analog_cnt
number of analog channels;

p_digital_cnt
number of digital channels

p_rpm_cnt
number of RPM channels

p_digio_cnt
number of digital-IO channels;
is always 0;

Return:

Number of LTT devices found.

2.1.2 LTTDeInit

[]

Declaration:

```
void __stdcall   LTTDeInit( void );
```

Description:
Shuts down internal systems and frees resources.

Input:

Return:

2.1.3 LTTConnectDevices

[Device]

Declaration:

```
void __stdcall LTTConnectDevices( void );
```

Description:

Connect to devices for exclusive use.

Input:

Return:

2.1.4 LTTReleaseDevices

[Device]

Declaration:

```
void __stdcall LTTReleaseDevices( void );
```

Description:

Closes open connections to devices.

Input:

Return:

2.1.5 LTTSetTransferMode

NEW

[Device]

Declaration:

```
int32_t __stdcall LTTSetTransferMode(
      int32_t type,
      const char *p_local,
      const char *p_netmask,
      const char *p_remote
);
```

Description:

Changes OSAL transfer subsystem.

IMPORTANT:
!Must be called before LTTInit() !

Input:

type
transfer subsystem
1 – direct access (default)
3 – NETROA over network
4 – LTTROA system service

p_local
pointer to c-string containing IP-address of local network adapter;

p_netmask
pointer to c-string containing IP-netmask of local network adapter;

p_remote
pointer to c-string containing comma separated list of IP-addresses of remote LTTROAD-system to which LTT2API should connect to;

Return:

0 - Always

2.1.6 LTTSetDeviceOrder

NEW

[Device]

Declaration:

```
int32_t __stdcall LTTSetDeviceOrder(
      uint32_t dev_cnt,
      const uint32_t *p_devorder
);
```

Description:

Sets a device order.
If device-order list doesn't match the devices found, it will be ignored.

IMPORTANT:
Must be called before LTTInit() !

Input:

dev_cnt
: length of device-order list;
[0..3]

p_devorder
: pointer to list with serial numbers;

Return:

0 – OK
-1 – invalide pointer
-2 – device-count out-of-range

2.1.7 LTTNetRoa_Scan

NEW

[Device]

Declaration:

```
void __stdcall LTTNetRoa_Scan(
     const char *p_local,
     const char *p_netmask
);
```

Description:

Scans network for available NETROA clients with LTT-devices

Input:

p_local

pointer to c-string containing IP-address of local network adapter;

p_netmask

pointer to c-string containing IP-netmask of local network adapter;

Return:

number of clients found.

2.1.8 LTTNetRoa_GetScanAddress

NEW

[Device]

Declaration:

```
int32_t __stdcall LTTNetRoa_GetScanAddress(
      uint32_t scan_idx,
      char *p_addr
);
```

Description:

Gets IP-address of client **scan_idx** from a previous LTTNetRoa_Scan() call..

Input:

scan_idx

index; must be smaller than the number of clients found

p_netmask

pointer to c-string where IP-address will be stored;

IMPORTANT:
Application must provide at least 32 Bytes !

Return:

0 – OK

2.2 Parameter Handling and Upload

2.2.1 LTTResetParameter

Changed

[Device]

Declaration:

```
void __stdcall LTTResetParameter(
     int32_t device_reset
);
```

Description:

Resets all parameter settings in LTT2API to default. When device_reset is 0 no upload to the device(s) occurs and only the DLL parameters are affected. If, however, device_reset equals 1 the channels are set to internal ground, that is OFF.

Input:

device_reset

option to reset the devices

0 – not passed; only API is reset

other – additionally resets devices;

Return:

2.2.2 LTTGetParameters

Changed

[Device]

Declaration:

```
int32_t __stdcall LTTGetParameters(
      LTTParam *p_para
);
```

Description:

Gets current valid setup.

Input:

p_para

Pointer to data-structure, where current setup parameters will be stored.

Return:

0	successful
-1	invalid pointer

2.2.3 LTTCascadingMode NEW

[]

Declaration:

```
void __stdcall LTTCascadingMode(
      int32_t mode
);
```

Description:

Specifies how multiple LTT devices will work together.
Independently or in cascading mode.

Input:

mode

specifies wether multiple LTT devices are running cascadied or independent, but still with the same timebase

0 - independent mode
1 - cascading mode

Return:

None

2.2.4 LTTMasterCorrectionMode NEW

[]

Declaration:

```
void __stdcall LTTMasterCorrectionMode(
     int32_t mode
);
```

Description:

Specifies how multiple LTT devices will work together. Independently or in cascading mode.

Input:

mode

switch correction on/off

0 - OFF
1 - ON

Return:

None

2.2.5 LTTSyncCorrection NEW

[]

Declaration:

```
void __stdcall LTTSyncCorrection(
     int32_t corr
);
```

Description:

Specifies how multiple LTT devices will work together.
Independently or in cascading mode.

Input:

corr

specifies wether multiple LTT devices are running cascadied or independent, but still with the same timebase

0 - independent mode
1 - cascading mode

Return:

None

2.2.6 LTTUploadParameters

Changed

[Device]

Declaration:

int32_t __stdcall LTTUploadParameters(void);

Description:

Checks and sends the parameters to the LTT devices.

The parameters have been gathered using LTTAnalogChannel, LTTAnalogDMSChannel, LTTSampleTime, LTTRAMBufferSetting, etc. but the device has not yet been configured.
LTTUploadParameters makes a final check to see if the combination of all the parameters don't make an error and then if successful configures the devices. The error codes are additive.

Input:

Return:

The return value is a sum of the following error codes.

0	successful
-1	LTT device not found or ASPI dll not found
-2	Wrong sample rate by current channel setting
-4	No channels active
-8	transfer active
-16	Wrong channel combination!!
-32	Trigger channel not active
-64	pre-trigger settings wrong - no channel set
-256	pre-trigger settings wrong - invalid pre-trigger length
-512	No Buffer settings specified!!
-1024	First device is not active (internal error)
-2048	filter cut-off frequency out of range

2.3 Channel Settings

2.3.1 LTTAnalogChannel **Changed**

[]

Declaration:

```
int32_t __stdcall LTTAnalogChannel(
      uint32_t ch_idx,
      uint32_t active,
      uint32_t range,
      uint32_t coup,
      uint32_t supply,
      uint32_t offset
);
```

Description:

Turns on/off analog channels on standard LTT hardware and apply the channel settings.

Input:

ch_idx

channel ID

active

0 - channel OFF
1 - channel ON
2 - channel ON - but not calibrated raw data

range

channel range in [mV]
Allowed settings:
Depends on the LTT device

coup

channel coupling as coded value [uint32_t]
Allow values:
Depends on the particular LTT device

supply

channel supply [mV,uA,mHz]

offset

channel offset [internal format, e.g. 0]

Return:

0	successful
-1	channel doesn't exist
-2	invalid coup
-3	coup doesn't match LTT-Hardware
-4	channel range doesn't match coup settings

2.3.2 LTTDigitalChannel

Changed

[]

Declaration:

```
int32_t __stdcall LTTDigitalChannel(
      uint32_t ch_idx,
      uint32_t active
);
```

Description:

Turns on/off digital channels.

Input:

ch_idx

channel ID, which to be modified.

active

0 - channel OFF
1 - channel ON

Return:

0	successful
-1	channel doesn't exist.

2.3.3 LTTRPMChannel

Changed

[]

Declaration:

```
int32_t __stdcall LTTRPMChannel(
      uint32_t ch_idx,
      uint32_t active
);
```

Description:

Turns on/off RPM channels.

CAUTION:
RPM-channels are only available at LTT-184/186 hardware and are deprecated !!

Input:

ch_idx
: channel ID, which to be modified.
Must be 0 !!

active
: 0 - channel OFF
1 - channel ON

Return:

0	successful
-1	channel doesn't exist.

2.3.4 LTT24PulseRecognitionChannel

NEW

[Device]

Declaration:

```
int32_t __stdcall LTT24PulseRecognitionChannel(
      uint32_t ch_id,
      uint32_t mode,
      uint32_t pls_cnt,
      uint32_t pls_ok_cnt,
      uint32_t pls_rot,
      uint32_t pls_mean,
      const float *p_corr
);
```

Description:

Configures the internal pulse recognition engine of channel.
LTT24 devices only!!

Input:

ch_id

Analog channel index

mode

<to be described> [0x0000...0x3FFF]

pls_cnt

<to be described> [0x0000...0x3FFF]

pls_ok_cnt

<to be described> [0x0000...0x3FFF]

pls_rot

<to be described> [0x0000...0x3FFF]

pls_mean

<to be described> [0x0000...0x3FFF]

p_corr

Pointer to pulse count correction buffer
correction factor is relativ error in the range [0.0...2.0].
no correction: [1.0]
max. 3072 values!

Return:

<to be described>

2.3.5 LTT24PlsRecogChannelAdvanced

NEW

[Device]

Declaration:

```
int32_t __stdcall LTTAnalogAutoZero(
      uint32_t ch_id,
      uint32_t gn_dphi,
      uint32_t gn_phid,
      uint32_t gn_phi2d,
      uint32_t gn_dphid,
      uint32_t gn_dphi2d
);
```

Description:

Configures the gains of the internal pulse recognition engine of channel.
LTT24 devices only!!

Input:

ch_id

Analog channel index

gn_dphi

<to be described> [0x0000...0x3FFF]

gn_phid

<to be described> [0x0000...0x3FFF]

gn_phi2d

<to be described> [0x0000...0x3FFF]

gn_dphid

<to be described> [0x0000...0x3FFF]

gn_dphi2d

<to be described> [0x0000...0x3FFF]

Return:

<to be described>

2.3.6 LTT24DACOutputChannel

NEW

[Device]

Declaration:

```
int32_t __stdcall LTTAnalogAutoZero(
      uint32_t ch_id,
      uint32_t line,
      uint32_t mode,
      uint32_t gain
);
```

Description:

Configures the analog output (DAC-Output) of channel.
LTT24 devices only!!

Input:

ch_id

Analog channel index

line

DAC-line at channel [0]
If pulse-recognition is active at channel then [0,1]

mode

DAC operation mode:
[LTT_CHNL_24_DACOUT_OFF,

no pulse recognition active:
LTT_CHNL_24_DACOUT_ADC,
LTT_CHNL_24_DACOUT_CIRCULAR_BUFF,
LTT_CHNL_24_DACOUT_STREAM_INTHD,

pulse recognition active:
LTT_CHNL_24_DACOUT_PULSE_DELTA_PHI_2_DOT,
LTT_CHNL_24_DACOUT_PULSE_DELTA_PHI_DOT,
LTT_CHNL_24_DACOUT_PULSE_DELTA_PHI,
LTT_CHNL_24_DACOUT_PULSE_PHI_2_DOT,
LTT_CHNL_24_DACOUT_PULSE_PHI_DOT,
LTT_CHNL_24_DACOUT_PULSE_PHI,
LTT_CHNL_24_DACOUT_PULSE_PULSE_COUNT]

gain

DAC gain [0...15]

Return:

<to be described>

2.3.7 LTT24DACOutputSignalChannel

NEW

[Device]

Declaration:

```
int32_t __stdcall LTTAnalogAutoZero(
      uint32_t ch_id,
      uint32_t line,
      uint32_t mode,
      uint32_t dac_tb,
      uint32_t sig_tb,
      float amp,
      float off,
      float phi0,
      const int16_t *p_buf
);
```

Description:

Configures the circular buffer of the analog output (DAC-Output) of channel.
LTT24 devices only!!

Input:

ch_id
: Analog channel index

line
: DAC-line at channel [0]

mode
: signal type:
[LTT_CHNL_24_DACOUT_SIG_DC,
LTT_CHNL_24_DACOUT_SIG_RECTANGLE,
LTT_CHNL_24_DACOUT_SIG_SINE,
LTT_CHNL_24_DACOUT_SIG_TRIANGLE,
LTT_CHNL_24_DACOUT_SIG_BUFFER,
LTT_CHNL_24_DACOUT_SIG_SAWTOOTH]

dac_tb
: dac output rate [ns]
[500, 1000, 2000, 4000, 8000]

sig_tb
: signal timebase [ns]

amp
: signal amplitude [relative to +/-5V]

off
: signal offset [relative to +/-5V]

phi0
: signal phase offset [radiant]

p_corr
: Pointer to user-defined buffer [int16_t]
Currently max. 1536 values!

Return:

<to be described>

2.3.8 LTT24DigitalLineChannel

NEW

[Device]

Declaration:

```
int32_t __stdcall LTT24DigitalLineChannel(
      uint32_t ch_id,
      uint32_t line,
      uint32_t coup,
      uint32_t filt,
      uint32_t mid,
      int32_t level
);
```

Description:

Configures the digital input lines track A,B und Z for pulse-recognition engine of channel. If pulse-recognition is not active only the information of track A will be used for digital channel.
LTT24 devices only!!.

Input:

ch_id

Analog channel index

line

Digital line at channel
[0 – Track A,
1 – Track B,
2 – Track Z]

coup

Digital line input coupling
[LTT_CHNL_24_COUP_DIGLINE_OFF,
LTT_CHNL_24_COUP_DIGLINE_DC,
LTT_CHNL_24_COUP_DIGLINE_AC]

filt

Digital line input filter frequence
[LTT_CHNL_24_FILT_DIGLINE_9MHZ,
LTT_CHNL_24_FILT_DIGLINE_16MHZ,
LTT_CHNL_24_FILT_DIGLINE_35MHZ,
LTT_CHNL_24_FILT_DIGLINE_190MHZ]

mid

Mid level for input range
[40 – for 0..80V,
35 – for -5..75V,
30 – for -10..70V,
25 – for -15..65V,
20 – for -20..60V,
15 – for -25..55V,
10 – for -30..50V]

level

Level in input range for high-low decision [mV]

Return:

<to be described>

2.3.9 LTTAnalogChargeClear

NEW

[Device]

Declaration:

```
int32_t __stdcall LTTAnalogChargeClear(
      uint32_t chnl_cnt,
      uint32_t *p_chnl_list
);
```

Description:

Charge-Clear for selected channels.

Input:

chnl_cnt

length of channel-list

p_chnl_list

pointer to channel-list for charge-clear

Return:

<to be described>

2.3.10 LTTAnalogAutoZero

Changed

[Device]

Declaration:

```
int32_t __stdcall LTTAnalogAutoZero(
      uint32_t chnl_cnt,
      uint32_t *p_chnl_list
);
```

Description:

Automatic ReZero of GND-Level for selected active channels in current range.
LTT24 and LTT180/182 devices only!

Input:

chnl_cnt

length of channel-list

p_chnl_list

pointer to channel-list for rezero'ing

Return:

<to be described>

2.3.11 LTTAnalogIOCompensation NEW

[Device]

Declaration:

```
int32_t __stdcall LTTAnalogIOCompensation(
      uint32_t chnl_cnt,
      uint32_t *p_chnl_list
);
```

Description:

Automatic ReZero of GND-Level for selected active channels in current range.
LTT184/186 devices only!

Input:

chnl_cnt

length of channel-list

p_chnl_list

pointer to channel-list for rezero'ing

Return:

<to be described>

2.3.12 LTTOutputImpedance

Changed

[]

Declaration:

```
int32_t __stdcall LTTOutputImpedance(
      uint32_t ch_idx,
      uint32_t conn_type,
      uint32_t impedance
);
```

Description:

Sets output impedance of signal source for specified input coupling.
LTT184/186 devices only!

Input:

ch_idx

Analog channel ID

conn_type

type of input coupling

0 - SE+ coupling
1 - SE- coupling
2 - DE coupling

impedance

impedance value in [Ohm];
values > 1000000000 --> infinite

Return:

0	successful
-1	channel doesn't exist.
-2	invalid connection type.

2.3.13 LTTSetInternReference

Changed

[]

Declaration:

```
int32_t __stdcall LTTSetInternReference(
      uint32_t dwDevice,
      uint32_t dwRefIdx
);
```

Description:

Sets all the channels for a given device to the internal reference.
LTT184/186 devices only!

Input:

dwDevice

Selects the LTT device.

dwRefIdx

index of reference value table

0 <=> GND
1 <=> 857mV
2 <=> 3.00V
3 <=> V_{temp}
4 <=> V_{temp} – 3.00V
5 <=> 3.00V – 3.00V
6 <=> 0 - 857mV

Return:

0	successful
-1	invalid device index
-2	invalid reference index

2.3.14 LTTGetChannelConverter

Changed

[]

Declaration:

```
double __stdcall LTTGetChannelConverter(
      uint32_t dwAnalogChannel
);
```

Description:

Gives back conversion factor for requested analog channel.
Applying this factor to a ADC data sample value will converts the value in units of [mV]:

[mV] = [ADC] * LTTGetChannelConverter([channel])

Input:

dwAnalogChannel

Analog channel index

Return:

conversion factor

2.3.15 LTTGetChannelRange

Changed

[]

Declaration:

```
int32_t __stdcall LTTGetChannelRange(
      uint32_t dwAnalogChannel
);
```

Description:

Gives back range for the requested channel.

Input:

dwAnalogChannel

Analog channel index

Return:

maximum range; +/- value in [mV]

2.3.16 LTTGetDMSChannelOffset

Changed

[]

Declaration:

```
int32_t __stdcall LTTGetDMSChannelOffset(
      uint32_t ch_idx
);
```

Description:

Obtains offset of DMS-channel

Input:

ch_idx

Channel index

Return:

coded channel offset

2.3.17 LTTGetChannelCouplingListLength

Changed

[]

Declaration:

```
int32_t __stdcall LTTGetChannelCouplingListLength(
      uint32_t dev_idx
      uint32_t coup
);
```

Description:

Gives back length of list of supported channel-ranges for given Input-Coupling

Input:

dev_idx

Device index.

coup

Input-Coupling

Return:

Length of list

2.3.18 LTTGetChannelCouplingList

Changed

[]

Declaration:

```
int32_t __stdcall LTTGetChannelCouplingList(
      uint32_t *p_rnglist,
      uint32_t dev_idx,
      uint32_t coup
);
```

Description:

fills list with supported channel ranges for given input-coupling

Input:

p_rnglist

Pointer to list, where list of supported ranges will be stored.

dev_idx

Device index.

coup

Input-Coupling

Return:

0	successful
-1	invalid pointer

2.3.19 LTTGetChannelRangeListLength

Changed

[]

Declaration:

```
int32_t __stdcall LTTGetChannelRangeListLength(
      uint32_t dev_idx
      uint32_t coup
);
```

Description:

Gives back length of list of supported channel-ranges for given Input-Coupling

Input:

dev_idx

Device index.

coup

Input-Coupling

Return:

Length of list

2.3.20 LTTGetChannelRangeList

Changed

[]

Declaration:

```
int32_t __stdcall LTTGetChannelRangeList(
      uint32_t *p_rnglist,
      uint32_t dev_idx,
      uint32_t coup
);
```

Description:

fills list with supported channel ranges for given input-coupling

Input:

p_rnglist

Pointer to list, where list of supported ranges will be stored.

dev_idx

Device index.

coup

Input-Coupling

Return:

0	successful
-1	invalid pointer

2.3.21 LTTGetChannelID NEW

[]

Declaration:

```
int32_t __stdcall LTTGetChannelID(
      uint32_t dev_idx
      uint32_t coup
);
```

Description:

Gives back length of list of supported channel-ranges for given Input-Coupling

Input:

dev_idx

Device index.

coup

Input-Coupling

Return:

Length of list

2.4 Timebase and Filter Settings

2.4.1 LTTSampleTime **Changed**

[]

Declaration:

```
int32_t __stdcall LTTSampleTime(
      uint32_t SampTime,
      int32_t TimeFlag
);
```

Description:

Sets the sampling time.

Input:

SampTime

Sample time in nano-seconds.

CAUTION:
not all times are allowed!

Supported sample times depend on the particular hardware type.
You can get a full list of supported sample-times from the API.

TimeFlag

If sample time is between two allowed sample times, this parameter indicates, which of both is chosen for legal sample time. Chooses the sampling time below (1) or above (0)

Return:

The actual sampling time used in [ns].

2.4.2 LTTFilter

Changed

[]

Declaration:

```
int32_t __stdcall LTTFilter(
      uint32_t filter_mode,
      uint32_t filter_cutoff
);
```

Description:

Sets the digital filter. A common filter for all channels is supported.

Input:

filter_mode

A combination of certain flags, which specifies the requested digital filter.

Filter mode:

[LTT_FILTER_OFF,
LTT_FILTER_ON]

Filter types:

[LTT_FILT_TYPE_BUTTERWORTH,
LTT_FILT_TYPE_BESSEL,
LTT_FILT_TYPE_CHEBYCHEV]

Filter pol:

[LTT_FILT_POL_TWO,
LTT_FILT_POL_FOUR,
LTT_FILT_POL_SIX,
LTT_FILT_POL_EIGHT,
LTT_FILT_POL_TEN,
LTT_FILT_POL_TWELVE,
LTT_FILT_POL_FOURTEEN,
LTT_FILT_POL_SIXTEEN]

filter_cutoff

Cut-Off frequency of the filter [ns] (-3dB point)

Return:

0	successful
-1	wrong filter type
-2	wrong filter pol

2.4.3 LTTGetResolution

Changed

[]

Declaration:

```
int32_t __stdcall LTTGetResolution( void );
```

Description:

Gets the current bit resolution for the sampling time.

Input:

Return:

12 12Bit resolution;
14 14Bit resolution;
15 15Bit resolution;
16. 16Bit resolution;

2.4.4 LTTGetSampleTimeListLength

Changed

[]

Declaration:

```
int32_t __stdcall LTTGetSampleTimeListLength( void );
```

Description:

gives back length of list of supported sample-times

Input:

Return:

length of list

2.4.5 LTTGetSampleTimeList

Changed

[]

Declaration:

```
int32_t __stdcall LTTGetSampleTimeList(
      int32_t *lpList
);
```

Description:

fills list with supported sample-times

Input:

lpList

Pointer to list, where list of supported sample times will be stored.

Return:

0	successful
-1	invalid pointer

2.5 Trigger Settings

2.5.1 LTTTrigger **Changed**

[]

Declaration:

```
int32_t __stdcall LTTTrigger(
      uint32_t Trigger,
      uint32_t Slope,
      uint32_t Channel,
      uint32_t ChType,
      uint32_t TrigType,
      float f1,
      float f2
);
```

Description:

Enables or disables the trigger channel and sets trigger parameters.

CAUTION:
The analog channels which will be used as trigger channels must be turned on!

Triggering on a digital channel does not require the digital channel is turned on
- firmware 6xxx
  - f1 - digital trigger bit [8..15]
- firmware 7xxx
  - f1 - digital trigger bit [10..15]
- CAUTION:
  Only the specified bits are allowed !!!

Input:

Trig
- Trigger state:
  [LTT_TRIG_STATE_OFF,
  LTT_TRIG_STATE_ON]

Slope

LTT_TRIG_MODE_LEVEL,
LTT_TRIG_MODE_LEVEL_DELTA,
LTT_TRIG_DIGMODE_SINGLE_BIT:
- [LTT_TRIG_SLOPE_POS,
  LTT_TRIG_SLOPE_NEG]

LTT_TRIG_MODE_COMPARISION:
- [LTT_TRIG_COMP_BIGGER,
  LTT_TRIG_COMP_SMALLER]

LTT_TRIG_MODE_REGION:
- [LTT_TRIG_REGION_ENTER,
  LTT_TRIG_REGION_EXIT]

Channel
- set trigger channel (index begins with 0)

ChType
Trigger channel type :
[LTT_CHNL_TYPE_ANALOG,
LTT_CHNL_TYPE_DIGITAL]

TrigType
Analog:
[LTT_TRIG_MODE_LEVEL,
LTT_TRIG_MODE_COMPARISION,
LTT_TRIG_MODE_REGION,
LTT_TRIG_MODE_LEVEL_DELTA]
Digital:
0 - single bit mode

f1
LTT_TRIG_MODE_LEVEL,
LTT_TRIG_MODE_COMPARISION,
LTT_TRIG_MODE_LEVEL_DELTA:
Threshhold [% FSR]

LTT_TRIG_MODE_REGION:
Lower Threshhold [% FSR]

LTT_TRIG_DIGMODE_SINGLE_BIT:
Trigger-Bit

f2
LTT_TRIG_MODE_LEVEL:
Sensitivity [% FSR]

LTT_TRIG_MODE_REGION:
Upper Threshhold [% FSR]

LTT_TRIG_MODE_LEVEL_DELTA:
minimal difference to previous sample [% FSR]

LTT_TRIG_MODE_COMPARISION,
LTT_TRIG_DIGMODE_SINGLE_BIT:
not used

Return:

0	successful
-1	channel doesn't exist.
-2	level trigger: can't have negative hysterisis.
-3	delta trigger: invalid range for f2 (0 < f2 < 6)
-4	level+delta trigger: invalid range for f2 (0 < f2).
-5	invalid digital trigger bit. FW 6xx (0 or 7 < Channel < 16) FW >700 (0 or 9 < Channel < 16)
-6	invalid trigger channel type
-7	invalid trigger type

2.5.2 LTTTrigger_SetPreTrigger

Changed

[]

Declaration:

```
int32_t __stdcall LTTTrigger_SetPreTrigger(
      uint32_t pre_triglen,
      uint32_t pre_sampletime
);
```

Description:

Sets the pre-trigger length per channel. Units are in kilo samples.

For example, desired is a 500ms pre-trigger using a 400ns sampling rate. The pre-trigger length is calculated.

500 000 000 ns (1 Sample / 400 ns) = 125 000 samples
125 000 samples / (1024 samples) = 122.070

Must be integer number so either 122 (499.7 ms) or 123 (503.8ms) are valid

pre-trigger is only allowed under certain conditions. Per devices only one or a pair number of active channels are allowed.

The pre-trigger length is limited to 60 Mega samples (120*1024*1024 byte).
So with 6 active channels the maximum pre-trigger per channel is 10 Mega samples.

Input:

pre_triglen:
Amount of pre-tigger samples [kS]

pre_sampletime:
Sample time in [ns] before trigger has happened.

CAUTION:
Not all times are allowed.

0 - use no different sampletime

Return:

0	successful
-1	No trigger active
-2	Wrong pre-trigger length

2.5.3 LTTTrigger_GetPreTriggerLimit NEW

[]

Declaration:

```
int32_t __stdcall LTTTrigger_GetPreTriggerLimit(
      uint32_t dwTime
);
```

Description:

Sets reactivation time for Trigger. Typical dead-time for reactivation <5µs.
Units are in [ns]. The timer resolution is in steps of 100ns.
The valid range depends on the used sample time:

0	- Reactivation timer OFF
4 * Sampletime...1s	- Valid reactivation time

CAUTION:
This option can't be activated together with LTTTrigger_SetPreTrigger!!
Both are mutually exclusive.

Input:

dwTime
reactivation time in [ns]

Return:

0	successful
-1	No trigger active
-2	Reactivation time too big

2.5.4 LTTTrigger_SetReactivationTime

Changed

[]

Declaration:

```
int32_t __stdcall LTTTrigger_SetReactivationTime(
      uint32_t dwTime
);
```

Description:

Sets reactivation time for Trigger. Typical dead-time for reactivation <5µs.
Units are in [ns]. The timer resolution is in steps of 100ns.
The valid range depends on the used sample time:

0	- Reactivation timer OFF
4 * Sampletime...1s	- Valid reactivation time

CAUTION:
This option can't be activated together with LTTTrigger_SetPreTrigger!!
Both are mutually exclusive.

Input:

dwTime
reactivation time in [ns]

Return:

0	successful
-1	No trigger active
-2	Reactivation time too big

2.5.5 LTTTrigger_SetABAMode NEW

[]

Declaration:

```
int32_t __stdcall LTTTrigger_SetABAMode(
      uint32_t dwTime
);
```

Description:

Sets reactivation time for Trigger. Typical dead-time for reactivation <5µs.
Units are in [ns]. The timer resolution is in steps of 100ns.
The valid range depends on the used sample time:

0	- Reactivation timer OFF
4 * Sampletime...1s	- Valid reactivation time

CAUTION:
This option can't be activated together with LTTTrigger_SetPreTrigger!!
Both are mutually exclusive.

Input:

dwTime
reactivation time in [ns]

Return:

0	successful
-1	No trigger active
-2	Reactivation time too big

2.5.6 LTTTrigger_TriggerScan NEW

[]

Declaration:

```
int32_t __stdcall LTTTrigger_TriggerScan(
      uint32_t dwTime
);
```

Description:

Sets reactivation time for Trigger. Typical dead-time for reactivation <5µs.
Units are in [ns]. The timer resolution is in steps of 100ns.
The valid range depends on the used sample time:

0	- Reactivation timer OFF
4 * Sampletime...1s	- Valid reactivation time

CAUTION:
This option can't be activated together with LTTTrigger_SetPreTrigger!!
Both are mutually exclusive.

Input:

dwTime
reactivation time in [ns]

Return:

0	successful
-1	No trigger active
-2	Reactivation time too big

2.6 Transfer Buffer Settings

2.6.1 LTTGetRAMBufferSettings **Changed**

[]

Declaration:

```
int32_t __stdcall LTTGetRAMBufferSettings(
      uint32_t dwSamplesPerChannel ,
      uint32_t dwChannels
);
```

Description:

Gives back needed buffer size in bytes.

This is a helper function to calculate the required buffer size for a certain channel settings and data amount.

Input:

dwSamplesPerChannel
: Number samples per channels

dwChannels
: Number of channels

Return:

The size of the buffer in bytes.

2.6.2 LTTRAMBufferSettings

Changed

[]

Declaration:

```
int32_t __stdcall LTTRAMBufferSettings(
      short *pwBuffer,
      uint32_t dwSamplesPerChannel,
      uint32_t dwChannels
);
```

Description:

Setting up the RAM buffer system for data transfer.

The buffer must be allocated by the application. The size must match the channel settings and data amount.

Transfer handling function is LTTTransfer2Buffers.

Input:

pwBuffer
: Pointer to RAM buffer.

dwSamplesPerChannel
: Number of samples per channel.

dwChannels
: Number of channels.

Return:

0	successful
-1	STREAM already set up !! RAM and STREAM are exclusive buffer settings.
-2	length per channel is invalid

2.6.3 LTTSTREAMSettings

Changed

[]

Declaration:

```
int32_t __stdcall LTTSTREAMSettings(
      short *pwBuffer,
      uint32_t dwBufferSize,
      uint32_t dwSamplesPerChannel,
      uint32_t dwChannels
);
```

Description:

Deprecated!

Is not possible to activate anymore.

Input:

pwBuffer
: Pointer to RAM-Buffer.

dwBufferSize
: Buffer size in samples

dwSamplesPerChannel
: Number of samples per channel

dwChannels
: Number of channels

Return:

-1 always

2.6.4 LTTSTREAMTransfer

Changed

[]

Declaration:

```
int32_t __stdcall LTTSTREAMTransfer(
      int32_t mode,
      uint32_t dwSamplesPerChannel
);
```

Description:

Activates new stream transfer.

Transfer handling function is LTTStream2Buffer.

Input:

mode

Buffer size in samples

dwSamplesPerChannel

Number of samples per channel

Return:

0	successful
-1	ERROR; RAM buffer already set

2.6.5 LTTBufferReset

Changed

[]

Declaration:

```
void __stdcall LTTBufferReset( void );
```

Description:

Resets the transfer buffers systems.

Input:

Return:

2.7 Transfer Control

2.7.1 LTTStart

Changed

[Device]

Declaration:

```
int32_t __stdcall LTTStart( void );
```

Description:

Starts the LTT devices and activates the trigger if on.

Input:

Return:

0	successful
-1	no device present or no active channels

2.7.2 LTTStop

[]

Declaration:

```
void __stdcall LTTStop( void );
```

Description:

Stops the data acquisition at LTT devices.

Input:

Return:

2.7.3 LTTCheckTrigger

Changed

[]

Declaration:

```
int32_t __stdcall  LTTCheckTrigger(
      uint32_t MilliSecond
);
```

Description:

Checks if the trigger has been fulfilled.

In the case the trigger is still active, the call waits the specified amount of time before the call returns.

Input:

MilliSecond

time in milliseconds to wait after a negative trigger result.

Return:

0 trigger not fulfilled

1 trigger successful

2.7.4 LTTTransfer2Buffers

Changed

[Device]

Declaration:

```
int32_t __stdcall LTTTransfer2Buffers(
      uint32_t buffer_type,
      uint32_t xfer_mode
);
```

Description:

Transfers the data from the devices to the assigned buffers.
This call initiates the real transfer. The transfer can work in BLOCKED or NONE_BLOCKED mode.

BLOCKED:

The call returns, when the transfer is done.
This is not possible with the STREAM buffer system.

NONE_BLOCKED:

The transfer is initiated and the call returns immediately.
The Transfer must be monitored with LTTDataTransferStatus.

Input:

buffer_type

Reserved; Always 0.

xfer_mode

0 - Blocked mode;
Call returns, when data are in buffer.
1 - NON-Blocked mode;
Call initiate transfer and returns immediately.
End of transfer can be obtained with LTTDataTransferStatus()

Some flags can be applied to the mode:

4 - Continuous transfer;
For subsequent calls of LTTTransfer2Buffers() a restart of the LTT-systems is not done. This is necessary for infinite-type of data acquisition.
8 - High precision mode;
Not possible with Continuous-Mode;
Reduces noise feed-back of SCSI-transfer.

Return:

-1 No LTT devices.
-2 No buffer type set.
-3 Transfer is active, so can't invoke another transfer !!
-4 File transfer already done. Reconfigure file buffer settings!

BLOCKED mode:
>=0 Number of transferred samples

CAUTION:
The amount of transferred samples must be checked against the requested data length. If the amount of transferred samples is less than the requested amount, an irrecoverable error has occurred during the transfer, which led to a break of the transfer. In the data buffer the correctly transferred data are stored up to the point where the error has occurred!!

NONE_BLOCKED mode:
0 transfer invoked

2.7.5 LTTStream2Buffer

[Device]

Declaration:

```
int32_t __stdcall LTTStream2Buffer(
      short *p_buffer
      uint32_t seq_cnt,
      uint32_t xfer_mode
);
```

Description:

Transfers the data from the devices to the submitted buffer.
This call initiates the real transfer. The transfer can work in BLOCKED or NONE_BLOCKED mode.

BLOCKED:
The call returns, when the transfer is done.
This is not possible with the STREAM buffer system.

NONE_BLOCKED:
The transfer is initiated and the call returns imediately.
The Transfer must be monitored with LTTDataTransferStatus.

Input:

p_buffer:
Pointer to RAM-Buffer.

seq_cnt:
amount of sequences to store; must be multiple of 1024

xfer_mode:
0 - Blocked mode;
Call returns, when data are in buffer.
1 - NON-Blocked mode;
Call initiate transfer and returns imediately.
End of transfer can be obtained with 'LTTDataTransferStatus'

Return:

-1	No LTT devices.
-2	STREAM transfer not active
-3	Transfer is active, so can't invoke another transfer !!
-4	Sequence count is not valid

BLOCKED mode:

>=0	Number of transferred samples

CAUTION:
The amount of transferred samples must be checked against the requested data length. If the amount of transferred samples is less than the requested amount, an irrecoverable error has occurred during the transfer, which led to a break of the transfer. In the data buffer the correctly transferred data are stored up to the point where the error has occurred!!

NONE_BLOCKED mode:

0	transfer invoked

2.7.6 LTTStream2BufferCB

[Device]

Declaration:

```
int32_t __stdcall LTTStream2BufferCB(
      short *p_buffer
      uint32_t seq_cnt,
      uint32_t xfer_mode
);
```

Description:

Transfers the data from the devices to the submitted buffer.
This call initiates the real transfer. The transfer can work in BLOCKED or NONE_BLOCKED mode.

BLOCKED:
The call returns, when the transfer is done.
This is not possible with the STREAM buffer system.

NONE_BLOCKED:
The transfer is initiated and the call returns imediately.
The Transfer must be monitored with LTTDataTransferStatus.

Input:

p_buffer:
Pointer to RAM-Buffer.

seq_cnt:
amount of sequences to store; must be multiple of 1024

xfer_mode:
0 - Blocked mode;
Call returns, when data are in buffer.
1 - NON-Blocked mode;
Call initiate transfer and returns imediately.
End of transfer can be obtained with 'LTTDataTransferStatus'

Return:

-1 No LTT devices.
-2 STREAM transfer not active
-3 Transfer is active, so can't invoke another transfer !!
-4 Sequence count is not valid

BLOCKED mode:
>=0 Number of transferred samples

CAUTION:
The amount of transferred samples must be checked against the requested data length. If the amount of transferred samples is less than the requested amount, an irrecoverable error has occurred during the transfer, which led to a break of the transfer. In the data buffer the correctly transferred data are stored up to the point where the error has occurred!!

NONE_BLOCKED mode:
1 transfer invoked

2.7.7 LTTDataTransferStatus

[]

Declaration:

```
int32_t __stdcall LTTDataTransferStatus( void );
```

Description:

Monitors the status of the transfer.

This function is necessary for the **NONE_BLOCKED** Mode of transfer.

Input:

Return:

0	nothing TODO
-1	waiting for trigger
-2	new block available
-3	no transfer active
-255	error happened
>0	transfer complete, number of transmitted samples

CAUTION:
The amount of transferred samples must be checked against the requested data length. If the amount of transferred samples is less than the requested amount, an irrecoverable error has occurred during the transfer, which led to a break of the transfer. In the data buffer the correctly transferred data are stored up to the point where the error has occurred!!

2.7.8 LTTDataTransferWait

[]

Declaration:

```
int32_t __stdcall LTTDataTransferWait( void );
```

Description:

Waits until the transfer has finished.

This function is necessary for the **NONE_BLOCKED** Mode of transfer.

Input:

Return:

-3	no transfer active
-255	error happened
>0	transfer complete, number of transmitted samples

CAUTION:
The amount of transferred samples must be checked against the requested data length. If the amount of transferred samples is less than the requested amount, an irrecoverable error has occurred during the transfer, which led to a break of the transfer. In the data buffer the correctly transferred data are stored up to the point where the error has occurred!!

2.7.9 LTTDataTransferBreak

[Device]

Declaration:

```
int32_t __stdcall LTTDataTransferBreak( void );
```

Description:

Breaks a running transfer.

Input:

Return:

0	successful
-1	error occurred

2.7.10 LTTData_GetCounter

[]

Declaration:

```
int32_t __stdcall LTTData_GetCounter( void );
```

Description:

Gives back the amount of transferred samples.

This call can be used in **NONE_BLOCKED** Mode of transfer to monitor the progress of the transfer.

Input:

Return:

amount of up to now transferred samples

2.7.11 LTTDataTransferCounter

[]

Declaration:

```
int32_t __stdcall LTTDataTransferCounter( void );
```

Description:

Gives back the amount of transferred samples.

This call can be used in **NONE_BLOCKED** Mode of transfer to monitor the progress of the transfer.

Input:

Return:

amount of up to now transferred samples

2.7.12 LTTGetChannelSequenceLength

[]

Declaration:

```
int32_t __stdcall LTTGetChannelSequenceLength( void );
```

Description:

Gives back length of channel sequence.

Input:

Return:

Length of channel sequence.

2.7.13 LTTGetChannelIDSequence

[]

Declaration:

```
void __stdcall LTTGetChannelIDSequence(
     uint32_t *pSequenceList
);
```

Description:

Gives back channel ID in channel sequence.

Input:

pSequenceList

pointer to list. Application must provide sufficient memory !!

Return:

None

2.8 Internal HD Access

2.8.1 LTTHDGetSectorCount

[]

Declaration:

```
int32_t __stdcall LTTHDGetSectorCount(
      uint32_t *p_dev_sec_cnt,
      uint32_t info_size
);
```

Description:
<to be described>

Input:
p_dev_sec_cnt
<to be described>

info_size:
<to be described>

Return:
<to be described>

2.8.2 LTTHDRecordReplay

[]

Declaration:

```
int32_t __stdcall LTTHDRecordReplay(
      uint32_t rec_idx,
      char *p_info,
      uint32_t info_max_size
);
```

Description:
<to be described>

Input:
rec_idx
<to be described>

p_info
<to be described>

info_max_size
<to be described>

Return:
<to be described>

2.8.3 LTTHDRecordReplaySetPos

[]

Declaration:

```
int32_t __stdcall LTTHDRecordReplaySetPos(
      uint64_t seq_pos
);
```

Description:
<to be described>

Input:
seq_pos
<to be described>

Return:
<to be described>

2.8.4 LTTHDRecordLeaveReplay

[]

Declaration:

```
int32_t __stdcall LTTHDRecordLeaveReplay( void );
```

Description:
<to be described>

Input:
None

Return:
<to be described>

2.8.5 LTTHDRecordLength

[]

Declaration:

```
int32_t __stdcall LTTHDRecordLength(
      uint32_t dwHDSamplesPerChannel
);
```

Description:
<to be described>

Input:
dwHDSamplesPerChannel
<to be described>

Return:
<to be described>

2.8.6 LTTHDRecordButton

[]

Declaration:

```
int32_t __stdcall LTTHDRecordButton(
      uint32_t use_button
);
```

Description:
<to be described>

Input:
use_button
<to be described>

Return:
<to be described>

2.8.7 LTTHDGetRecordInfo

[]

Declaration:

```
int32_t __stdcall LTTHDGetRecordInfo(
      LTTRecordInfo *p_recinfo
);
```

Description:
<to be described>

Input:
p_recinfo
<to be described>

Return:
<to be described>

2.8.8 LTTHDGetRecordParameter

[]

Declaration:

```
int32_t __stdcall LTTHDGetRecordParameter(
      uint32_t rec_idx,
      LTTParam *p_para,
      char *p_info,
      uint32_t info_max_size
);
```

Description:
<to be described>

Input:
rec_idx
<to be described>

p_para
<to be described>

p_info
<to be described>

info_max_size
<to be described>

Return:
<to be described>

2.9 Miscellaneous

2.9.1 LTTGetCalibTemperature

[]

Declaration:

```
int32_t __stdcall LTTGetCalibTemperature(
      uint32_t dwDevice
);
```

Description:

Gives back internal temperature of specified device at last calibration.

Input:

dwDevice

Device ID for which the calibration temperature is requested

Return:

Internal devices temperature in degrees C

2.9.2 LTTGetTemperature

[Device]

Declaration:

```
int32_t __stdcall LTTGetTemperature(
      uint32_t device
);
```

Description:

Measure temperature in specified device.

Input:

device

Device ID for which the calibration temperature is requested

Return:

current internal devices temperature

2.9.3 LTTGetDLLInfo

[]

Declaration:

```
int32_t __stdcall LTTGetDLLInfo(
      char *pcBuffer
);
```

Description:

Obtain identification string of the DLL.

Input:

pcBuffer

pointer to buffer where the null terminated string will be put.
Must be at least 64 bytes.

Return:

0 always

2.9.4 LTTGetVersion

[]

Declaration:

```
int32_t __stdcall LTTGetDLLVersion( void );
```

Description:

Obtain version number of the DLL.
The version number is build in the following manner:

(LTT2API_MAJOR << 16) | (LTT2API_MINOR << 8) | LTT2API_BUG

Input:

Return:

Version number

2.9.5 LTTGetDeviceInfo

[]

Declaration:

```
int32_t __stdcall LTTGetDeviceInfo(
      uint32_t dwDevice,
      char *pcBuffer
);
```

Description:

Obtain identification string of the LTT device.

Input:

dwDevice

Selects the LTT device.

pcBuffer

pointer to buffer where the null terminated string will be put.
Must be at least 64 bytes.
Example:

```
LTT-184/16 SN:3214 FW:6.0.607 ch:(   1 -  16)
```

Return:

0	successful
-1	device not found.

2.9.6 LTTGetDeviceInfoEx

[]

Declaration:

```
int32_t __stdcall LTTGetDeviceInfoEx(
      uint32_t dwDevice,
      LTTDeviceInfo *spDevInfo
);
```

Description:

Obtain identification string of the LTT device.

Input:

dwDevice

Selects the LTT device.

spDevInfo

pointer to buffer where the null terminated string will be put.
Must be at least 64 bytes.

Return:

0	successful
-1	device not found.

2.9.7 LTTGetDevices

[]

Declaration:

```
int32_t __stdcall LTTGetDevices( void );
```

Description:
Returns the number of devices found.

Input:

Return:
number of devices found.

2.9.8 LTTChannel2Device

[]

Declaration:

```
int32_t __stdcall LTTChannel2Device(
      uint32_t dwChannel,
      uint32_t dwChannelType
);
```

Description:

Returns device index of channel.

Input:

dwChannel:

channel index

dwChannelType:

channel type

Return:

>=0	device index
-1	invalid channel type
-2	invalid channel index

2.9.9 LTTChannelAtDevice

[]

Declaration:

```
int32_t __stdcall LTTChannelAtDevice(
      uint32_t dwChannel,
      uint32_t dwChannelType
);
```

Description:

Returns channel index at device of channel

Input:

dwChannel:

channel index

dwChannelType:

channel type

Return:

>=0	device channel index
-1	invalid channel type
-2	invalid channel index

2.9.10 LTTDevice2Channel

[]

Declaration:

```
int32_t __stdcall LTTDevice2Channel(
      uint32_t dwDevice,
      uint32_t dwDevChannel,
      uint32_t dwChannelType
);
```

Description:

Returns channel index of device index and device channel index

Input:

dwDevice:

device index

dwDevChannel:

device channel index

dwChannelType:

channel type

Return:

>=0	channel index
-1	invalid channel type
-2	invalid device index
-3	invalid device channel index