LLT.DLL

Interface documentation

MICRO-EPSILON Optronic GmbH
Lessingstr. 14
01465 Dresden-Langebrueck

Interface documentation on LLT.dll

The LLT.dll is a DLL for the simple integration of the scanCONTROL into own applications. It builds a level of abstraction over the direct respond of the scanCONTROL by Ethernet or the serial interface. In the design of this DLL a special value has been set on the simplicity of the interface and a high performance.
For using this DLL with as much different application development systems and compilers as possible, the DDL-interface has been realised with pure C-functions of the “cdecl” and “stdcall” call convention. Thus the DLL may also be used under C, Delphi or other programming languages (the compatibility of the used file types is condition for this). For C++ applications an additional class exists by which the C-functions are mapped by methods of an interface-class.

In this documentation only the integration of the DLL in C++ is described, the integration in C or another programming language may be derived from this documentation.

ScanCONTROL devices with Ethernet connection are monitored with heart beat packets. In order to avoid connection aborts during application debugging, the heart beat time out should be increased (see chapter 2.9.9 “Ethernet Heartbeat timeout”).

1. Loading the DLL

There are two different possibilities for loading a DLL. It may be loaded directly when an application is started (static) or later if required dynamically. The dynamic loading is mostly more favourable, particularly because of providing a better error processing.

1.1. Static loading

When starting a DLL in a C or C++ project statically, the corresponding *.lib-file is compiled into the project with the definitions of the DLL-functions. Thereby it is important always to use the .lib-file compatible to the used DLL-version. In the header files `C_InterfaceLLT_2.h` and `S_InterfaceLLT_2.h` the corresponding function declarations are to be found. There the prefix `s_` stands for “stdcall” and `c_` for “cdecl”.

For other programming languages the import function for the DLL may be generated with the `C_InterfaceLLT_2.h` or the `S_InterfaceLLT_2.h`.

In case of a new DLL-version the project has to be recompiled as the entry points in the DLL may change.

1.2. Dynamic loading

For the loading of the LLT.dll and the import of its functions in C++-projects the two classes `CInterfaceLLT` and `CDllLoader` are provided.
The `DllLoader` is responsible for the DLL handling (loading and respond of the function pointer). In the interface class (`CInterfaceLLT`) all functions exported by the LLT.dll are defined as function pointer. Therefore they simply may be called as methods of the interface class.

```
#include "InterfaceLLT_2.h"
CInterfaceLLT* pInterfaceLLT;

...

//creating the interface-class
pInterfaceLLT = new CInterfaceLLT();

//testing if the requested function is available
if(pInterfaceLLT->m_pFunctions->CreateLLTDevice != NULL)
{
  //call of the functions in the LLT.dll
  pInterfaceLLT->CreateLLTDevice(INTF_TYPE_ETHERNET);
}
```

The outstanding advantage of this interface class is the dynamic call of the DLL functions. This means, it may be inserted a new DLL version without recompiling the project. Additionally it may be polled, if the requested function is available in the DLL. But this is only necessary for the functions added after the first release.
If new functions of the LLT.dll are to be used, the project has to be recompiled with the actual interface class.

Additionally the constructor of the interface class may be passed the name of the DLL to be loaded (with path) and a pointer to a `bool`-variable, which signalises an error when loading the DLL.

2. Description of the single functions

The functions of the LLT.dll are divided into several functional groups:

Functional group	Description
Instance functions	For creating a scanCONTROL-instance with Ethernet or serial interface support and for deleting this instance
Chooser functions	For choosing a scanCONTROL
Connection functions	Connecting and closing of a connection with a scanCONTROL
Identification functions	Function for interrogating the name and the version
Property functions	Function for interrogating and setting properties (e.g. shutter time, serial number…)
Special property functions	Function for interrogating and setting special properties
Register functions	For registering a callback and an error message
Profile transfer functions	Functions for the transfer of profiles
IS functions	For interrogating the different states and connections
Partial profile functions	Restriction of the profile to be transmitted on the scanCONTROL
Time functions	Evaluation of the timestamp
Post processing functions	Reading and writing of the post processing parameter
File functions	Loading and saving of profiles
Special CMM trigger functions	Starting and stopping of the profile transfer including the CMM triggers
Error value conversion functions	Conversion of the error value of functions in text

For all functions described here always the parameters for the `CInterfaceLLT`-class are given. If this class is not used, every function has an additional first parameter (`Instanzhandle`). All other parameter of the functions are moved one backwards (see file `C_InterfaceLLT_2.h` or `S_InterfaceLLT_2.h`).
This additional first parameter serves as differentiation of the different scanCONTROL instances in the DLL.
In the `CInterfaceLLT` class this parameter is added automatically.

2.1. Functions of the LLT.dll up to Version 3.0.0

The new instance functions “CreateLLTDevice”, “IsInterfaceType” and “GetInterfaceType” were added to support the new type of connection, namely Ethernet. Since these functions have to be uniformly used for all types of connection, the previous functions have to be considered as being obsolete. They solely serve for the downward compatibility of the LLT.dll.

Obsolete name of the function	**New name of the function**	**Chapter**
`CreateLLTSerial`	`CreateLLTDevice(INTF_TYPE_SERIAL)`	2.3
`IsSerial`	`IsInterfaceType(INTF_TYPE_SERIAL)`	2.12

2.2. General return values of the functions

All functions of the interface return an `int` value as return value. If the return value of a function is greater than or equal to `GENERAL_FUNCTION_OK` respectively `'1'`, so the function has been successful, if the return value `GENERAL_FUNCTION_NOT_AVAILABLE` respectively `'0'` or negative, so an error occurred.
There is a specialty at some functions which may also return `GENERAL_FUNCTION_CONTAINER_MODE_HEIGHT_CHANGED` respectively. `'2'`. If this return value appears the size of the image in the container mode has changed (see chapter 2.11 “Profile transfer functions”).

For the differentiation of the single return values several constants are available. In the following table all general return values, which may be returned by functions are listed. For the single functional groups additionally there may also be special return / error values.

Constant for the return value	Value	Description
`GENERAL_FUNCTION_CONTAINER_MODE_HEIGHT_CHANGED`	2	Function successfully executed, but the image size for the container mode has changed
`GENERAL_FUNCTION_OK`	1	Function successfully executed
`GENERAL_FUNCTION_NOT_AVAILABLE`	0	This function is not available, possibly using a new DLL or switching to the Ethernet mode
`ERROR_GENERAL_WHILE_LOAD_PROFILE`	-1000	Function could not be executed as the loading of profiles is active
`ERROR_GENERAL_NOT_CONNECTED`	-1001	There is no connection to the scanCONTROL -> call `Connect`
`ERROR_GENERAL_DEVICE_BUSY`	-1002	The connection to the scanCONTROL is interfered or disconnected -> reconnect and check interface of the scanCONTROL
`ERROR_GENERAL_WHILE_LOAD_PROFILE_OR_GET_PROFILES`	-1003	Function could not be executed as either the loading of profiles or the profile transfer is active
`ERROR_GENERAL_WHILE_GET_PROFILES`	-1004	Function could not be executed as the profile transfer is active
`ERROR_GENERAL_GET_SET_ADDRESS`	-1005	The address could not be read or written. Possibly a too old firmware is used
`ERROR_GENERAL_POINTER_MISSING`	-1006	A required pointer is set `NULL`
`ERROR_GENERAL_WHILE_SAVE_PROFILES`	-1007	Function could not be executed as the saving of profiles is active
`ERROR_GENERAL_SECOND_CONNECTION_TO_LLT`	-1008	A second instance must be connected to this scanCONTROL via Ethernet or the serial port. Please close the second instance.

2.3. *Instance functions*

The LLT.dll supports the communication with the scanCONTROL via a serial interface or Ethernet. After loading of the DLL, an appropriate device has to be created in the DLL by means of `CreateLLTDevice(InterfaceType)`. In case of success, these functions return `GENERAL_FUNCTION_OK`.
The InterfaceType that is used for the creation of a device can be retrieved via the function GetInterfaceType ().

The following InterfaceType values are supported:

Constants for InterfaceType	**Value**	**Description**
`INTF_TYPE_UNKNOWN`	0	If it is returned by GetInterfaceType in case of error, it is impermissible for CreateLLTDevice.
`INTF_TYPE_SERIAL`	1	A connection via the serial interface
`INTF_TYPE_ETHERNET`	3	A connection via Ethernet

If the `CInterfaceLLT` class is not used these two functions return an `Instanzhandle` for the internal instance created in the LLT.dll instead of `GENERAL_FUNCTION_OK` or `GENERAL_FUNCTION_NOT_AVAILABLE`. This handle has to be passed as first parameter to all further functions.
If this is `Instanzhandle 0` or `0xffffffff` the creation of a device has failed.

By the function `DelDevice()` the created device has to be deleted before the unload of the DLL (this is made automatically in the `CInterfaceLLT` class).

In a `DelDevice()` all parameter set up in the scanCONTROL remain preserved, except the “Profile config” (see chapter 2.9.5), the “Packet size” (see chapter 2.9.4) and the “Buffer count” of the driver (see chapter 2.9.1).

2.4. *Parallel operating of several scanCONTROL*

There are two possibilities, depending on the chosen method for loading the LLT.dll for using several scanCONTROL parallel in one programme.

Static loading:

Per connected scanCONTROL appropriately the function `CreateLLTDevice()`has to be called. By different returned `Instanzhandle` the different scanCONTROL respectively instances may be distinguished.

Dynamic loading:

Per connected scanCONTROL an appropriate instance of the `CInterfaceLLT` class has to be created. The instances of the class administrate independently the `Instanzhandle` for every scanCONTROL.

If callbacks are to be used the different instances may share one callback (see chapter 2.10.1 “Callback”).

```
#include <vector>

void main()
{
  pInterfaceLLT_1 = new CInterfaceLLT();
  pInterfaceLLT_2 = new CInterfaceLLT();

  //creation of two ethernet devices/instances
  pInterfaceLLT_1->CreateLLTDevice(INTF_TYPE_ETHERNET);
  pInterfaceLLT_2->CreateLLTDevice(INTF_TYPE_ETHERNET);

  std::vector<unsigned int> EthernetInterfaces(6);

  //readout of the available interfaces
  Int InterfaceCount = pInterfaceLLT_1->GetDeviceInterfaces(
    & EthernetInterfaces[0], EthernetInterfaces.size());

  if(InterfaceCount < 2)
  {
    //only 1 scanCONTROL is connected
    return;
  }

  //setting of the different interfaces
  pInterfaceLLT_1->SetDeviceInterface(EthernetInterfaces[0], 0);
  pInterfaceLLT_2->SetDeviceInterface(EthernetInterfaces[1], 0);

  //connection of two scanCONTROL
  pInterfaceLLT_1->Connect();
  pInterfaceLLT_2->Connect();

  //registry of the Callback
  pInterfaceLLT_1->RegisterCallback(STD_CALL, (void*)NewProfile, 0));
  pInterfaceLLT_2->RegisterCallback(STD_CALL, (void*)NewProfile, 1));
  . . .
}

//callback
void _stdcall NewProfile (const unsigned char* Data, unsigned int Size,
                           void* UserData)
{
  switch((int)UserData)
  {
    case 0:
    {
      //callback called by LLT_1
    }
    case 1:
    {
      //callback called by LLT_2
    }
  }
}
```

2.5. *Choosing functions*

By the choosing functions a selection of the device interfaces may be effected. For example the used serial interface or the IP address may be chosen.

```
int GetDeviceInterfaces(unsigned int *pInterfaces, unsigned int nSize)
int GetDeviceInterfacesFast(unsigned int *pInterfaces, unsigned int nSize)
```

Call for the device interfaces to which a scanCONTROL is connected. The `Interfaces` parameter is a field of integer variables in which the interface numbers are entered. `Size` here specifies the size of the field.
In the communication by a serial interface the numbers of all useable com-ports (Port 1 to 16), associated to a scanCONTROL are entered into the field, whereas in case of an Ethernet connection, the IP addresses of all devices found.
The return value returns the number of found and in the `Interfaces` field entered device interfaces.
The fast-function gives the answer quicker at Ethernet connection, but in large networks it can happen that not all sensors found.
If the return value is less than or equal to `GENERAL_FUNCTION_NOT_AVAILABLE` it may be one of the general or following return values (see chapter 2.2 “General return values of the functions”):

Constant for the return values	**Value**	**Description**
ERROR_GETDEVINTERFACES_WIN_NOT_SUPPORTED	-250	Function is only available for Windows 2000 or higher
ERROR_GETDEVINTERFACES_REQUEST_COUNT	-251	The size of the passed field is to small
ERROR_GETDEVINTERFACES_INTERNAL	-253	A error occurred during the scanCONTROL enumeration

```
void SetDeviceInterface(unsigned int nInterface, int nAdditional)
```

Setting of the number of an interface and transfer of an additional parameter. The additional parameter is used in order to indicate the baud rate of the serial interface. Therefore, the value has to be 115 200. In case of Ethernet interface, nInterface corresponds to the IP address in the so-called host byte order, wherein the 4 bits of the IP address of the sequence are still saved from the highest to the lowest byte. This can be easily achieved by means of a shift operation of the individual bytes of the IP address, as seen in the example for the address 192.168.1.2:

```
(192<<24) | (168<<16) | (1<<8) | 2 = 3232235778 = 0xC0A80102
```

The additional parameter can be applied while using the Ethernet interface in order to indicate the local IP address of the computer, to which the sensor has to send the data. This can be required for solving the problems in computers with multiple network interface cards. Normally, 0 has to be indicated, then the LLT.DLL determines the optimum IP address by itself.

If the return value is less than or equal to `GENERAL_FUNCTION_NOT_AVAILABLE` it may be one of the general or following return values (see chapter 2.2 "General return values of the functions"):

Constant for the return values	**Value**	**Description**
`ERROR_GETDEVINTERFACES _CONNECTED`	-252	The scanCONTROL is connected, please call `Disconnect();`

```
void SetDiscoveryBroadcastTarget(unsigned int nNetworkAddress,
                                 unsigned int nSubnetMask)
```

Sets the local sender address that is used for Discovery Packets.

A Broadcast packet, to which all available devices respond, is sent in order to discover devices that are connected via an Ethernet interface. Normally, this happens via a limited Broadcast. This Broadcast that is directed to the IP address 255.255.255.255 is not routed in the network. Alternatively, this function can limit the Broadcast to a single network adapter. `nNetworkAddress` determines the locale IP-address used as origin. `nSubnetMask` must be 0. Use `SetDiscoveryBroadcastTarget(0, 0)` to switch back to the standard behaviour.

This function is only supported by Ethernet interface, other interface instances issue the return value `GENERAL_FUNCTION_NOT_AVAILABLE`.

```
unsigned int nAvailableInterfaces[20];

pInterfaceLLT->SetDiscoveryBroadcastTarget(
                (192<<24) | (168<<16) | (1<<8) | (1<<0),
                0,
                );
int nInterfaces = pInterfaceLLT->GetDeviceInterfaces
                                          (nAvailableInterfaces, 20);

if(nInterfaces > GENERAL_FUNCTION_NOT_AVAILABLE)
{
  pInterfaceLLT->SetDeviceInterface(nAvailableInterfaces[0], 0);
}
```

2.6. Connection functions

By the connection functions a connection to a selected scanCONTROL may be established or disassociated.

```
int Connect()
```

Connecting the LLT.dll with a selected scanCONTROL. If there has not been chosen a scanCONTROL by `SetDeviceInterface` before, at a serial connection the com port 1 is chosen. In case of Ethernet interface, there is no default IP address that could be connected. Thus, it is indispensible to indicate a valid IP address via SetDeviceInterface in advance.

The return value may be one of the general or the following return values (see chapter 2.2 “General return values of the functions”):

Constant for the return value	Value	Description
`ERROR_CONNECT_LLT_COUNT`	-300	There is no scanCONTROL connected to the computer or the driver is not installed correctly.
`ERROR_CONNECT_SELECTED_LLT`	-301	The selected interface is not available -> choose a new interface with `SetDeviceInterface`
`ERROR_CONNECT_ALREADY_CONNECTED`	-302	There is already a scanCONTROL connected with this ID
`ERROR_CONNECT_LLT_NUMBER_ALREADY_USED`	-303	The requested scanCONTROL is already used by another instance -> choose another scanCONTROL with `SetDeviceInterface`
`ERROR_CONNECT_SERIAL_CONNECTION`	-304	The scanCONTROL by serial interface could not be connected -> choose another scanCONTROL with `SetDeviceInterface`

```
int Disconnect()
```

Disconnection of a scanCONTROL.
In a disconnect all parameter set up in the scanCONTROL remain preserved, except the “Profile config” (see chapter 2.9.5), the “Packet size” (see chapter 2.9.4) and the “Buffer count” in the driver (see chapter 2.9.1).

2.7. Identification functions

Functions for identifying the scanCONTROL.

2.7.1. GetDeviceName

```
int GetDeviceName(char *pDevName, unsigned int nDevNameSize,
                  char *pVenName, unsigned int nVenNameSize)
```

Query the device name and the manufacturer name of the scanCONTROL. For example, the device name is “LLT2800-100(000)v17-C2” or “scanCONTROL 2700-100(000)v20-C2”. The measurement range (in this case 100 mm), the option (000) and the software version of the scanCONTROL are coded in the device name.

If one of these names is not required there can also be a `NULL` passed instead of a pointer to the buffer.

The return value may be one of the general ort he following return values (see chapter 2.2 “General return values of the functions”):

Constant for the return value	Value	Description
ERROR_GETDEVICENAME_SIZE_TOO_LOW	-1	The size of a buffer is to small
ERROR_GETDEVICENAME_NO_BUFFER	-2	No buffer has been passed

```
char DeviceName[100];

memset(DeviceName, 0, sizeof(DeviceName));
if(pInterfaceLLT->GetDeviceName(DeviceName, sizeof(DeviceName),
                            NULL, NULL) > GENERAL_FUNCTION_NOT_AVAILABLE)
{
  //processing of the device name
}
```

2.7.2. GetLLTVersions

```
int GetLLTVersions(unsigned int *pDSP, unsigned int *pFPGA1,
                   unsigned int *pFPGA2)
```

Call for the software versions of the scanCONTROL. They will be extracted automatically from the device name.

2.7.3. GetLLTType

```
int GetLLTType(TScannerType *pScannerType)
```

Query measurement range and type of the scanCONTROL. They are automatically extracted from the device name. If the value `StandardType` is returned, the type could not be extracted from the name. Please contact Micro-Epsilon in order to obtain a current DLL.

TScannerType	**Value**	**scanCONTROL Type**	**Measuring range**
StandardType	-1	-	-
scanCONTROL28xx_25	0	28xx	25 mm
scanCONTROL28xx_100	1	28xx	100 mm
scanCONTROL28xx_10	2	28xx	10 mm
scanCONTROL27xx_25	1000	27xx	25 mm
scanCONTROL27xx_100	1001	27xx	100 mm
scanCONTROL27xx_50	1002	27xx	50 mm
scanCONTROL26xx_25	2000	26xx	25 mm
scanCONTROL26xx_50	2002	26xx	50 mm
scanCONTROL26xx_100	2001	26xx	100 mm
scanCONTROL29xx_25	3000	29xx	25 mm
scanCONTROL29xx_50	3002	29xx	50 mm
scanCONTROL29xx_100	3001	29xx	100 mm

2.8. Property functions

With the property functions different properties of the scanCONTROL may be read or written. There is a `GetFeature` and a `SetFeature` function. By a parameter the property to be read or written may be selected.

```
int GetFeature(DWORD function, DWORD *pValue);
int SetFeature(DWORD function, DWORD Value);
```

The return value may be one of the general or following return values (see chapter 2.2 "General return values of the functions"):

Constant for the return value	Value	Description
ERROR_SETGETFUNCTIONS_WRONG _FEATURE_ADRESS	-155	The address of the selected property is wrong

In the following table all available properties are listed:

Constant for the property	Address	Description
FEATURE_FUNCTION_SERIAL	0xf0000410	Serial
FEATURE_FUNCTION_LASERPOWER	0xf0f00824	Laser power
FEATURE_FUNCTION_MEASURINGFIELD	0xf0f00880	Measuring field
FEATURE_FUNCTION_TRIGGER	0xf0f00830	Trigger
FEATURE_FUNCTION_SHUTTERTIME	0xf0f0081c	Shutter time
FEATURE_FUNCTION_IDLETIME	0xf0f00800	Idle time
FEATURE_FUNCTION_PROCESSING_PROFILEDATA	0xf0f00804	Processing profile data
FEATURE_FUNCTION_THRESHOLD	0xf0f00810	Threshold
FEATURE_FUNCTION_MAINTENANCEFUNCTIONS	0xf0f0088c	Maintenance functions
FEATURE_FUNCTION_ANALOGFREQUENCY	0xf0f00828	Analog frequency
FEATURE_FUNCTION_ANALOGOUTPUTMODES	0xf0f00820	Analog output modes
FEATURE_FUNCTION_CMMTRIGGER	0xf0f00888	CMM trigger
FEATURE_FUNCTION_REARRANGEMENT_PROFILE	0xf0f0080c	Rearrangement profile
FEATURE_FUNCTION_PROFILE_FILTER	0xf0f00818	Profile filter
FEATURE_FUNCTION_RS422_INTERFACE _FUNCTION	0xf0f008c0	RS422 Interface Funktion
FEATURE_FUNCTION_PACKET_DELAY	0x00000d08	Packet delay

Whether the individual features in the connected scanCONTROL are available can be interrogated using the Inquiry properties. These Inquiry properties return the general availability, the respective minimum and maximum adjustable value and the availability of automatic regulation.

Constants for the Inquiry property	Address	Description
INQUIRY_FUNCTION_LASERPOWER	0xf0f00524	Laser power
INQUIRY_FUNCTION_MEASURINGFIELD	0xf0f00580	Measuring field
INQUIRY_FUNCTION_SHUTTERTIME	0xf0f0051c	Shutter time
INQUIRY_FUNCTION_IDLETIME	0xf0f00500	Idle time
FEATURE_FUNCTION_PROCESSING_PROFILEDATA	0xf0f00504	Processing profile data
INQUIRY_FUNCTION_THRESHOLD	0xf0f00510	Threshold
INQUIRY_FUNCTION_MAINTENANCEFUNCTIONS	0xf0f0058c	Maintenance functions

`INQUIRY_FUNCTION_ANALOGFREQUENCY`	`0xf0f00528`	Analogue frequency
`INQUIRY_FUNCTION_ANALOGOUTPUTMODES`	`0xf0f00520`	Analogue output modes
`INQUIRY_FUNCTION_CMMTRIGGER`	`0xf0f00588`	CMM trigger
`INQUIRY_FUNCTION_REARRANGEMENT_PROFILE`	`0xf0f0050c`	Rearrangement profile
`INQUIRY_FUNCTION_PROFILE_FILTER`	`0xf0f00518`	Profile filter
`INQUIRY_FUNCTION_RS422_INTERFACE _FUNCTION`	`0xf0f005c0`	RS422 Interface Funktion

The value of the Inquiry properties can be interpreted using the following table. Properties which are not available cannot be read or specified. The value of a property must always be within the limit values read out using the Inquiry properties. Higher values are ignored and can result in malfunctions.

Bit	**Description**
11..0	Maximum adjustable value
23..12	Minimum adjustable value
25	Automatic regulation is available
31	Availability of the property

```
DWORD nValue;

//Query the Inquiry property for the Shutter time
if(pInterfaceLLT->GetFeature(INQUIRY_FUNCTION_SHUTTERTIME, &nValue)) ==
GENERAL_FUNCTION_OK)
{
  //decode the values
  unsigned int nMaxvalue = nValue & 0x00000fff;
  unsigned int nMinvalue = (nValue & 0x00fff000)>>12;
  bool bAutoMode = (bool)((nValue & 0x02000000) >> 25);
  bool bAvailable = (bool)((nValue & 0x80000000) >> 31);
}
```

2.8.1. Serial

Reading the serial number. This property may only be read.

2.8.2. Laser power

Interrogating or setting of the laser power. The laser can be switched off, switched on with reduced power or switched on with standard power. Depending on device family, the polarity of the safety interlock can be adjusted (detailed information for this property can be found in section “Laser Power” in “OpManPartB.html”).

The first profile after changing the laser power may be corrupt. Therefore it is useful to process only the second profile afterwards.

2.8.3. Measuring field

Interrogating or setting of the measuring field. The measuring field number may be between 0 and 95. For further information see chapter 8.2 “Measuring field selection and calibration” and 11.3 “Supported measuring fields” in the manual of the scanCONTROL.

The selection of the measuring field has influence on the maximum profile frequency of the scanCONTROL (see chapter “Maximum Frequencies of profile Measurements” in the “QuickReference.html”).

The first profile after changing the measurement field may be corrupt. Therefore it is useful to process only the second profile afterwards.

2.8.4. Trigger

Interrogating or setting the trigger. Devices may be triggered internally or by an external input (detailed information for this property can be found in section “External Trigger input” in “OpManPartB.html”).

The first profile after shifting the trigger mode may be corrupt. Therefore it is useful to process only the second profile afterwards.

2.8.5. Shutter time

Interrogating or setting the shutter time. It may be between 1 and 4095. A counting value corresponds here 10 µs. From the sum of shutter time and dead time (see chapter 2.8.6 “Idle time”) the profile frequency may be calculated.

The image sensor supports interleaved read out. When the shutter time for reading out the sensor is enough, the dead time may be reduced to 30 µs. The maximum profile rate depends on the resolution (number of points per profile) and the measuring field. More details can be found in Chapter 8.2 “Measuring field selection and calibration” and in Chapter 11.3 “Supported measuring fields” in the operating instructions. A table with the profile rates to be achieved for the different resolutions and packet sizes can be found in “QuickReference.html”.

profile rate = 1000 / (shutter time in ms + idle time in ms)

By the 24th bit additionally an automatic shutter time regulation may be switched on. It regulates the shutter time depending on the measuring object. The lower 12 bit are giving an default value, which is used if no measuring object is seen. In the automatic shutter time regulation the profile rate does not change.

2.8.6. Idle time

Interrogating and setting the idle time. It may be between 1 and 4095. A counting value corresponds here 10 µs. From the sum of dead time and shutter time (see chapter 2.8.5 “Shutter time”) the profile frequency may be calculated.

profile rate = 1000 / (shutter time in ms + idle time in ms)

If the automatic shutter time regulation is active (see chapter 2.8.5 “Shutter time”), the dead time remains automatically adapted for keeping the profile frequency stable.

2.8.7. Processing profile data

Interrogating or setting the settings for profile processing (detailed information for this property can be found in „Processing of Profile Data“ of „OpManPartB.html“).

2.8.8. Threshold

Threshold for the selection of reflections. At targets with several reflections this may lead to better results (detailed information for this property can be found in „Threshold“ of „OpManPartB.html“).

2.8.9. Maintenance functions

Interrogating or setting the internal settings. Most of these settings should be left at their default values (detailed information for this property can be found in "Maintenance functions" of "OpManPartB.html").
Only the bits for the encoder input may be changed according to application. They activate the internal encoder counter. Upon activating, the counter value is reset to 0. The actual value is captured for every profile and transmitted in the time stamp.

```
#include <vector>
//activating the internal counter, setting resolution
pInterfaceLLT->SetFeature(FEATURE_FUNCTION_MAINTENANCEFUNCTIONS,
                          0x00000010);
pInterfaceLLT->SetResolution(256);

//activating of the profile transfer and waiting a moment (for profiles)
pInterfaceLLT->TransferProfiles(NORMAL_TRANSFER, true);
Sleep(500);

//creating a buffer for a profile in PURE_PROFILE Mode and fetching
//of a profile
unsigned int ProfSize = 256*4+16;
std::vector<unsigned char> vProfile(ProfSize);
if(pInterfaceLLT->GetActualProfile(&vProfile[0], ProfSize, PURE_PROFILE,
                                   NULL) != ProfSize)
{
  //the scanCONTROL does not send profiles
  return;
}
unsigned int LastCount = 0, OverflowCount = 0, TempCount;

pInterfaceLLT->Timestamp2CmmTriggerAndInCounter(&vProfile[ProfSize-16],
                                          TempCount, NULL, NULL, NULL);

//test whether the 16 bit counter is overflown
if(TempCount < (LastCount & 0x0000ffff))
{
  OverflowCount += 1;
}
//creation of the real counter reading with all overflows
LastCount = TempCount + 0x00010000 * OverflowCount;
```

2.8.10. Analogue frequency

Interrogating and setting of the frequency for the analogue output. The frequency may be set between 0 and 150 whereat the counting value is equal to the frequency in kHz. At the setting of 0 kHz the analogue output will be turned off which is reasonable at profile frequencies higher than 500 Hz for avoiding an overflow in the analogue output.

This parameter is only available for scanCONTROL 28xx. Detailed information for this property can be found in section „Speed of Analogue Outputs“ of „OpManPartB.html“.

2.8.11. Analogue output modes

Setting of the analogue output modes. E.g. the voltage range and the polarity of the analogue outputs may be shifted.

This parameter is only available for scanCONTROL 28xx. Detailed information for this property can be found in section „ Analogue Output Mode“ of „OpManPartB.html“.

2.8.12. CMM trigger

Configuration of the optional CMM triggers. The configuration of the CMM triggers consists of 4 instruction words. These instruction words have to be written successively. In reading the CMM trigger out only the last written instruction word may be re-read.. Detailed information for this property can be found in section " cmmTrigger " of "OpManPartB.html".

2.8.13. Rearrangement profile

In the container mode several profiles are combined to one container / image. Forthermore the sensor can rearrange the data in a way that allows direct image processing. Detailed information for this property can be found in section "Container Mode for Transmission" of "OpManPartB.html".

The return value may be one of the general or following return values (see chapter 2.2 “General return values of the functions”):

Constant for the return value	Value	Description
`ERROR_SETGETFUNCTIONS _REARRANGEMENT_PROFILE`	-159	The `Rearrangement` parameter is wrong

The width of the image is to be calculated as follows:

width = number of properties (bit 0 to 8) * 2 * resolution * number of stripes

The numbers of properties and stripes refers only to the ones put out but not to the existing ones.
After writing this property the LLT.dll calculates independently the necessary width of the image and sets it up. The width may be read out as described in chapter 2.9.7 “Profile container size”.

The packet size for the transfer of the converted profiles always has to correspond to an integer multiple of the column width. During the start of the transfer the LLT.dll automatically creates the suiting packet size, which is conforming to the active adjusted packet size at most.

Alternatively also successive profiles may be connected by setting only the field "Points per profile". The function `SetProfileContainerSize(0, nProfileCount)` determines the number of profiles within one container in this cases.

2.8.14. Profile filter

By the profile filter simple filters may be applied to a profile directly in the scanCONTROL. Thereby e.g. irregular points may be sorted out (detailed information for this property can be found in section "Profile Filter" in "OpManPartB.html").

2.8.15. Interface function/Port configuration

Determines the functions of the digital port. The interface can be used for serial communication or external trigger, respectively. The setting of this register must be consistent with other settings. For example, to use the externel trigger the interface has to be changed to external trigger and the External Trigger Input has to be activated.

The interface has also an automatic mode (default). The interface powers up as serial interface (115200baud) and changes to external trigger mode as soon as the External Trigger Input is activated. The interface remains in external trigger mode until the interface function is changed manually or next power up.

Detailed information about the interface configuration can be found in sctions „Serial Interface Function" and „Multi-Function Port Configuration" in „OpManPartB.html".

2.8.16. Packet Delay

The packet delay parameter determines the maximum bandwidth used by the sensor. It is the minimum time in microseconds the sensor must wait after sending one data packet before it sends the next data packet. Allowed values are between 0 and 1000. Using a nonzero packet delay may be necessary to avoid data overflow if several sensors are connected to a switch and must share a single connection to the PC.

2.8.17. Peakfilter Width and Intensity

Filter restricts the properties of the located peaks (detailed information for this property can be found in section "Extra Parameter" in "OpManPartB.html").
`SetFeature(FEATURE_FUNCTION_SHARPNESS,0)`activate the new parameter.

```
// set peakfilter width to min=7, max=55
DWORD peakwidth = (55<<16)+7;
pInterfaceLLT->SetFeature(FEATURE_FUNCTION_PEAKFILTER_WIDTH, peakwidth);
pInterfaceLLT->SetFeature(FEATURE_FUNCTION_SHARPNESS, 0); // activated the
new setting
```

2.8.18. Free Measuring field

Set the parameter for the free measuring field (detailed information for this property can be found in section “Extra Parameter” in “OpManPartB.html”).
`SetFeature(FEATURE_FUNCTION_SHARPNESS,0)`activate the new parameter.

```
// set free measuringfield
// start x 29591
// size x 6553
DWORD field = (29591<<16) + 6553;
pInterfaceLLT->SetFeature(FEATURE_FUNCTION_FREE_MEASURINGFIELD_X, field);
pInterfaceLLT->SetFeature(FEATURE_FUNCTION_SHARPNESS, 0); // activated the
new setting
```

2.9. Special property functions

By the special property functions further properties of the scanCONTROL may be written or read which do not suit the concept of the standard property functions.

2.9.1. Buffer count

```
int GetBufferCount(DWORD *pValue)
int SetBufferCount(DWORD Value)
```

Interrogating and setting of the buffer count in the scanCONTROL driver. The higher the count of buffers the more profiles / container may be cached before they have to be fetched by the LLT.dll. An automatic buffer control for use with an external trigger can also be specified.

A high buffer count is particularly reasonable at high profile frequencies, for slow processors and/or processors at which several programmes are running in the background. If the buffer count is to low it amounts to breakdowns in the profile transfer so that the profile frequency decreases. By default 20 buffers are set.
If a transfer has to be started in the container mode it has to be seen, that the count of used buffers is 4 maximum. Otherwise it may amount to a very slow start of the profile transfer and very high memory consumption. If very large containers are used also 3 buffers are sufficient.

The return value may be one of the general or following return values (see chapter 2.2 “General return values of the functions”):

Constant for the return value	Value	Description
`ERROR_SETGETFUNCTIONS_WRONG_BUFFER_COUNT`	-150	Count of the required buffer does not lie in the range of >=2 and <= 200

2.9.2. Main reflection

```
int GetMainReflection(DWORD *pValue)
int SetMainReflection(DWORD Value)
```

Interrogation and setting of the stripe to be put out at a profile configuration of `PURE_PROFILE` or `QUARTER_PROFILE`. This index is needed in the LLT.dll for converting `PROFILE` to `PURE_PROFILE` and `QUARTER_PROFILE`.
The index of the stripes to be put out is from 0 for the 1st reflection to 3 for the 4th reflection.

A description of the profile data and the stripes is to be found in chapter 2.22 “Description of the profile data”

This function is only available for Ethernet transfer.

The return value may be one of the general or following return values (see chapter 2.2 “General return values of the functions”):

Constant for the return value	Value	Description
ERROR_SETGETFUNCTIONS_REFLECTION_NUMBER_TOO_HIGH	-154	The index of the stripe to be put out is greater than 3

2.9.3. Max file size

```
int GetMaxFileSize(DWORD *pValue)
int SetMaxFileSize(DWORD Value)
```

Interrogation and setting of the maximum file size in the saving of profiles in byte. If this size has been reached the saving stops.

2.9.4. Packet size

```
int GetMinMaxPacketSize(unsigned long *pMinPacketSize,
                        unsigned long *pMaxPacketSize)
```

Interrogation of the minimum and maximum packet sizes of the Ethernet streaming packets. This function is only available in a Ethernet transfer.

```
int GetPacketSize(DWORD *pValue)
int SetPacketSize(DWORD Value)
```

Interrogation and setting of the active packet size of the Ethernet streaming packets. This packet size has to be between the minimum and maximum packet size. scanCONTROL supports the packet sizes 128, 256, 512, 1024, 2048 and 4096 bytes. For Ethernet connections, packets larger than 1024 bytes require the support of jumbo frames by all devices, especially by the receiving network card.

The return value may be one of the general or following return values (see chapter 2.2 “General return values of the functions”):

Constant for the return value	Value	Description
ERROR_SETGETFUNCTIONS_PACKET_SIZE	-151	The requested packet size is not supported

```
unsigned long MaxPacketSize = 0;

if(pInterfaceLLT->GetMinMaxPacketSize(NULL, &MaxPacketSize) > 0)
{
  pInterfaceLLT->SetPacketSize(MaxPacketSize);
}
```

2.9.5. Profile config

```
int GetProfileConfig(TProfileConfig *pValue)
int SetProfileConfig(TprofileConfig Value)
```

Interrogation and setting of the profile configuration.

ProfileConfig	Value	Description
`PROFILE`	1	Profile data of all four stripes
`PURE_PROFILE`	2	Reduced profile data of one stripe (only position and distance values)
`QUARTER_PROFILE`	3	Profile data of one stripe
`PARTIAL_PROFILE`	5	Partial profile which has been restricted by `SetPartialProfile`
`CONTAINER`	1	Container data
`VIDEO_IMAGE`	1	Video image of the scanCONTROL

In a connection by the serial interface only the profile configuration `PURE_PROFILE` is supported.
The profile configuration at the same time is applied for the saving of profiles and the callback. Continuative see chapter 2.21 “Profile / container / video transfer”.

The return value may be one of the general or following return values (see chapter 2.2 “General return values of the functions”):

Constant for the return value	Value	Description
`ERROR_SETGETFUNCTIONS_WRONG _PROFILE_CONFIG`	-152	The requested profile configuration is not available

```
TProfileConfig ProfileConfig;

pInterfaceLLT->GetProfileConfig(&ProfileConfig);

if(ProfileConfig != PURE_PROFILE)
{
  ProfileConfig = PURE_PROFILE;
  pInterfaceLLT->SetProfileConfig(&ProfileConfig);
}
```

2.9.6. Resolution

```
int GetResolution(DWORD *pValue)
int SetResolution(DWORD Value)
```

Query or setting the resolution of profiles. The standard resolutions for the different scanCONTROL variants are shown in the following table:

The return value may be one of the general or following return values (see chapter 2.2 “General return values of the functions”):

Constant for the return value	Value	Description
`ERROR_SETGETFUNCTIONS_NOT_SUPPORTED_RESOLUTION`	-153	The requested resolution is not supported

All possible resolution may be read out by the function `GetResolutions`.

```
int GetResolutions(DWORD *pValue, unsigned int nSize)
```

To this function a field of DWORD variables has to be passed. In this field then all possible resolutions are entered. At the moment only 5 different resolutions maximum are possible. The size of the field is too small if `ERROR_ SETGETFUNCTIONS_SIZE_TO_LOW` is returned otherwise the number of the entered resolutions.

The return value may be one of the general or following return values (see chapter 2.2 “General return values of the functions”):

Constant for the return value	Value	Description
`ERROR_SETGETFUNCTIONS_SIZE_TOO_LOW`	-156	The size of the passed field is too small

The following applies for the scanCONTROL 28xx: The larger the resolution of the profile, the smaller is the maximum profile frequency (see Chapter “Maximum Frequencies of Profile Measurements” in “QuickReference.html” for the scanCONTROL2800).

The resolution may only be changed if no profiles are transferred. Furthermore all properties for the `PartialProfile` are deleted at `SetResolution` (see chapter 2.13 “Partial profile function”).

2.9.7. Profile container size

```
int GetMaxProfileContainerSize(unsigned int *pMaxWidth, unsigned int
*pMaxHeight)
```

Interrogation of the maximum `ProfileContainerSize` for the container mode. (see chapter 2.8.13 “Rearrangement profile”).
If the maximum width is 64, the container mode will not be supported by the scanCONTROL.

```
int GetProfileContainerSize(unsigned int *pWidth, unsigned int *pHeight)
int SetProfileContainerSize(unsigned int nWidth, unsigned int nHeight)
```

Interrogation and setting of the size of the container for the container mode.
The width is set up automatically by the call of `SetFeature(FEATURE_FUNCTION_REARRANGEMENT_PROFILE)` (see chapter 2.8.13 “Rearrangement profile”).
The height may be selected free between 0 and the height maximum possible and corresponds to the number of profiles transferred in the container.
The container height must not be higher as the treble profile rate (see chapter 2.8.5 “Shutter time”), as it has to be ensured, that at least every 3 seconds a container is transferred. Otherwise it may amount to substantial failures.

If not all parameters are necessary alternatively it may be passed also `NULL` for not required ones.

The return value may be one of the general or following return values (see chapter 2.2 “General return values of the functions”):

Constant for the return value	**Value**	**Description**
`ERROR_SETGETFUNCTIONS_WRONG_PROFILE_SIZE`	-157	The size for the container is wrong
`ERROR_SETGETFUNCTIONS_MOD_4`	-158	The container width is not divisible by 4

If the “connection of successive profiles” is activated (see chapter 2.8.13 “Rearrangement profile”), the height * width of an image has to be an integral multiple of 16384. If it is tried to set up another height value, the height will be adjusted automatically to the next matching value. Additionally the error value `GENERAL_FUNCTION_CONTAINER_MODE_HEIGHT_CHANGED` will be returned for indicating the changes.

2.9.8. Loading and storing the user mode

Loading and storing the user mode. All settings of a scanCONTROL can be stored in a user mode so that all settings become active again immediately after a reset or restart. This is mainly expedient for post processing applications. Loading the user mode cannot be performed during an active profile / container transmission.
User mode 0 can only be loaded as it contains the factory settings.

```
int GetActualUserMode(unsigned int *pActualUserMode,
                      unsigned int *pUserModeCount);
```

Query of the available user modes. `pActualUserMode` is a pointer to a field with integer values for the available user modes. `pUserModeCount` is the size of the transmitted field. The scanCONTROL 28xx support 4 and the scanCONTROL 26xx/27xx/29xx 16 user modes.

```
unsigned int UserModes[10];

//Query of the available user mode
if(pInterfaceLLT->GetActualUserMode(&UserModes, sizeof(ErrorString)) >
GENERAL_FUNCTION_NOT_AVAILABLE)
{
  //Evaluation of the user mode
}
```

```
int ReadWriteUserModes(int nWrite, unsigned int nUserMode);
```

Loading or storing a user mode. If `nWrite` = 0, the user mode specified by `nUserMode` is loaded, otherwise the current settings are stored for this user mode. After loading a user mode it is necessary to call disconnect and connect.

Constants for the return value	**Value**	**Description**
`ERROR_SETGETFUNCTIONS_USER _MODE_TOO_HIGH`	-160	The specified user mode number is not available
`ERROR_SETGETFUNCTIONS_USER _MODE_FACTORY_DEFAULT`	-161	User mode 0 cannot be overwritten (factory settings)

```
//Storing the user mode 1
if(pInterfaceLLT->ReadWriteUserModes(1, 1) >
GENERAL_FUNCTION_NOT_AVAILABLE)
{
  //User mode has been stored successfully
}

//Loading the user mode 1
if(pInterfaceLLT->ReadWriteUserModes(0, 1) >
GENERAL_FUNCTION_NOT_AVAILABLE)
{
  //User mode has been loaded successfully
}
```

```
int SaveGlobalParameter(void);
```

Only useful for ethernet sensors, because it saves the IP-Properties.

2.9.9. Ethernet Heartbeat timeout

```
int SetEthernetHeartbeatTimeout(DWORD Value)
int GetEthernetHeartbeatTimeout(DWORD *pValue)
```

Setting and reading the Heartbeat Timeout in milliseconds to monitor the connection between llt.dll and scanCONTROL device. This is the time between two monitoring packets. A connection abort will occur if the device does not receive packets after three times the set up value. The LLT.dll sends a message in case of connection abort (see chapter 2.10.2 "Message"). Allowed values are between 500 and 1000000000ms.

This function sis available for devices with ethernet interface only.

The return value may be one of the general or following return values (see chapter 2.2 "General return values of the functions"):

Constant for the return value	**Value**	**Description**
`ERROR_SETGETFUNCTIONS` _HEARTBEAT_TOO_HIGH	-162	parameter value too large

2.9.10. HoldBuffersForPolling

```
int GetHoldBuffersForPolling(unsigned int *puiHoldBuffersForPolling)
int SetHoldBuffersForPolling(unsigned int uiHoldBuffersForPolling)
```

Settting or reading the profile count, that the llt.dll may hold for GetActualProfile (see section 2.11.5 „Get actual profile“). A larger count lets the llt.dll hold more profiles before profiles will be dropped if GetActualProfile is not called often enough.

Bit	**Funktion**
7..0	Count of held profiles

The count must not be higher than half the total buffer count (see section 2.9.1 „uffer count“).

The return value may be one of the general or following return values (see chapter 2.2 “General return values of the functions”):

Konstante für den Rückgabewert	**Wert**	**Beschreibung**
`ERROR_SETGETFUNCTIONS_WRONG_BUFFER_COUNT`	-150	Count of the required buffer does not lie in the range of >=2 and <= 200

2.10. Register functions

Functions for the registry of a Callback for the profile and a message for error messages.

2.10.1. Callback

```
int RegisterCallback(TCallbackType CallbackType,
                     void *pLLTProfileCallback, void *pUserData)
```

Registering of a callback of the type `TNewProfile_s` or `TNewProfile_c`, which is always called when a new profile / container has been received. This callback only has to be registered if it is meant to be used.
By the parameter `CallbackType` the call convention of the callback is specified. Does the call convention not correspond to the call convention of the own callback function it may amount to program crash's.

Callback type	Value	Description
`STD_CALL`	0	The callback is working with `stdcall` (`TNewProfile_s`)
`C_DECL`	1	The callback is working with `cdecl` (`TNewProfile_c`)

The parameter `pUserData` serves for the differentiation which scanCONTROL instance has called the callback. E.g. the pointer may be passed to a class or number.
If this function is passed `NULL` as `pLLTProfileCallback` the callback will be deactivated again. Whereby it has to be seen, that the call convention here also has to be indicated correctly.

Further information to this topic are to be found in the chapters 2.11 "Profile transfer functions" and 2.21 "Profile / container / video transfer".

```
typedef void (_stdcall *TNewProfile_s)(const unsigned char *pData,
                                     unsigned int nSize, void *pUserData);
typedef void (_cdecl *TNewProfile_c)(const unsigned char *pData,
                                     unsigned int nSize, void *pUserData);
```

These callbacks are, after they have been registered, called after the receipt of a profile / container and have as parameter a pointer to the profile / container data, the corresponding size of a data field and an `pUserData` parameter. With the `pUserData` parameter it may be distinguished from which scanCONTROL the profile / container originates.

The callback is intended for the processing of profiles / containers with a high profile frequency. During the callback profiles / container may be copied in a buffer for a later or a processing synchrony or asynchrony to the callback. A processing during the callback is not recommendable as for the time the callback needs for processing the LLT.dll is not able to fetch new profiles / container from the driver. Possibly by this it may amount to profile / container failures.
The profile / container data in the buffer passed by the callback must not be changed.

The profile configuration of the pointer may be set up by the function `SetProfileConfig` and applies at the same time for the saving of profiles / containers (see chapter 2.9.5 "Profile config").

```
#include <vector>
HANDLE hProfileEvent;
//setting of a buffer for a profile of the maximum profile size
std::vector<unsigned char> ProfileBuffer(1024*64);

void GetProfileFromCallback()
{
  //creation of an event
  hProfileEvent = CreateEvent(NULL, true, false, "ProfileEvent");

  pInterfaceLLT->RegisterCallback(STD_CALL, (void*)NewProfile_s, NULL);

  //activation of the profile transfer
  if(pInterfaceLLT->TransferProfiles(NORMAL_TRANSFER, true)
                                         <= GENERAL_FUNCTION_NOT_AVAILABLE)
    return;

  //waiting for an event from the profile callback
  if(WaitForSingleObject(hProfileEvent, 1000) != WAIT_OBJECT_0)
  {
    //error during the waiting for the callback
    return;
  }
  //evaluation of the profile

  //deactivation of the profile transfer
  if(pInterfaceLLT->TransferProfiles(NORMAL_TRANSFER, false)
                                         <= GENERAL_FUNCTION_NOT_AVAILABLE)
    return;
}
void CALLBACK NewProfile_s(const unsigned char *pData, unsigned int nSize,
                           void *pUserData)
{
  if(ProfileBuffer.size() >= nSize)
  {
    //if the profile is smaller than the buffer size:
    //copy of the profile into the buffer
    memcpy(&ProfileBuffer[0], pData, nSize);
    SetEvent(hProfileEvent);
  }
}
```

2.10.2. Message

```
int RegisterErrorMsg(UINT Msg, HWND hWnd, WPARAM WParam)
```

Registry of an error message send in case of errors.

Error code in LPARAM	Value	Description
`ERROR_SERIAL_COMM`	1	Error during the serial data transfer. Possibly the profile frequency is too high.
`ERROR_SERIAL_LLT`	7	ScanCONTROL could not interpret the commando or a parameter out of the validity area has been send.
`ERROR_CONNECTIONLOST`	10	The connection to the scanCONTROL has been interrupted (scanCONTROL has been switched off, reset or the Ethernet cable has been removed). Please send a „`Disconnect`“ for being able to reconnect. This message is only send at a connection by Ethernet.
`ERROR_STOPSAVING`	100	The saving of the profiles is finished (maximum data size reached).

2.11. Profile transfer functions

Description of the functions for the profile transfer. See chapter 2.21 “Profile / container / video transfer” for more detailed information.

Samples for the transfer of profiles with different profile configurations and the conversion of the profile data in millimetre are to be found in chapter 3 “LLT2800Samples”.

2.11.1. Transfer profiles

```
int TransferProfiles(int TransferProfileType, int nEnable);
```

Function for starting or terminating the profile transfer.
The parameter `TransferProfileType` indicates whether a continuous or a demand-based transfer has to be activated.

Transfer profile type	Value	Description
`NORMAL_TRANSFER`	0	Activation of a continuous transfer of profiles
`SHOT_TRANSFER`	1	Activation of a demand-based transfer of profiles (the transfer is always activated by `MultiShot`)
`NORMAL_CONTAINER_MODE`	2	Activation of a continuous transfer in the container mode
`SHOT_CONTAINER_MODE`	3	Activation of a demand-based transfer in the container mode (the transfer is always activated by `MultiShot`)

The parameter `nEnable` indicates whether a transfer has to be started or terminated. After the first start of a transfer it may take up to 100 ms before the first profiles / container are arriving per callback or are fetched per `GetActualProfile`.
If a transfer is terminated, the function waits automatically till the driver has returned all buffers. This may take up to 20 seconds.

If a transfer will be started in the container mode (see chapter 2.8.13 “Rearrangement profile”)it has to be seen, that the number of used buffers is 4 maximum (see chapter 2.9.1 “uffer count”). Otherwise it may amount to a very slow start of the profile transfer and a high memory consumption. If very large containers are used also 3 buffers are sufficient.
Furthermore it has to be seen, that every 3 seconds a container has to be transferred (see chapter 2.8.5 “Shutter time” and chapter 2.9.7 “Profile container size”). Otherwise it may amount to substantial failures.

The return value is the size of a profile / container. If the size is less than or equal to `GENERAL_FUNCTION_NOT_AVAILABLE` it is one of the general or following return values (see chapter 2.2 “General return values of the functions”):

Constant for the return value	Value	Description
`ERROR_PROFTRANS_PACKET_SIZE_TOO_HIGH`	-107	The packet size is bigger than the available one -> adjust a smaller packet size by `SetPacketSize`
`ERROR_PROFTRANS_CREATE_BUFFERS`	-108	The buffer for the driver could not be created duly -> possibly restart the PC
`ERROR_PROFTRANS_WRONG_PACKET_SIZE_FOR_CONTAINER`	-109	No appropriate packet size for the selected container settings can be found. Please increase the packet size (see Chapter 2.9.4 Packet size on page 23 .

2.11.2. Transfer video stream

```
int TransferVideoStream(int TransferVideoType, int nEnable,
                        unsigned int *pWidth, unsigned int *pHeight)
```

Function for starting or terminating the video picture transfer.
The parameter `TransferVideoType` indicates the transfer mode.

TransferVideoType	Value	Description
`VIDEO_MODE_0`	0	Decreased picture from the matrix
`VIDEO_MODE_1`	1	Full picture from the matrix

The size of the transferred pictures from the matrix will be copied into the parameter `pWidth` and `pHeight`.

Note that the maximum video frame rate is 25 pictures per second. Adjust the values for `Shutter time` and `Idle time` (see chapter 2.8.5 “Shutter time” and 2.8.6 “Idle time”).

The Packet size must be 4096 for the scanCONTROL28xx and 2048 for the scanCONTROL 26xx/27xx29xx.

Video pictures are fetched per `GetActualProfile`. The `Callback` is not available.
Video pictures can be saved a bitmap (see chapter 2.16.1 “aving of profiles”).

The return value is the size of video picture. If the size is less than or equal to `GENERAL_FUNCTION_NOT_AVAILABLE` it is one of the general or following return values (see chapter 2.2 “General return values of the functions”):

Constant for the return value	**Value**	**Description**
`ERROR_PROFTRANS_PACKET_SIZE_TOO_HIGH`	-107	The packet size is bigger than the available one -> adjust a smaller packet size by `SetPacketSize`
`ERROR_PROFTRANS_CREATE_BUFFERS`	-108	The buffer for the driver could not be created duly -> possibly restart the PC

2.11.3. Multi shot

By the functions `MultiShot` it is possible to transfer only one profile / container or a certain count of profiles / containers.
This feature is not available for the scanCONTROL 28xx until from DSP firmware version 10.

```
int MultiShot(unsigned int nCount)
```

Interrogation of several profiles / containers. The count of the profiles / container is passed in the parameter `nCount`. It may be interrogated between 1 and 65535 profiles / container.
If 0 is transmitted, the internal counter for the still outstanding profiles/container will be reset to 0 -> a new request can be made.

In order to be able to use MultiShot, the `SHOT_TRANSFER_MODE` or the `SHOT_CONTAINER_MODE` mode must have been selected for starting the transmission.

The return value may be one of the general or following return values (see chapter 2.2 “General return values of the functions”):

Constant for the return value	**Value**	**Description**
`ERROR_PROFTRANS_SHOTS_NOT_ACTIVE`	-100	The `SHOT_TRANSFER` mode or the `SHOT_CONTAINER_MODE` mode is not activated -> restart profile transfer
`ERROR_PROFTRANS_SHOTS_COUNT_TOO_HIGH`	-101	The count of interrogated profiles / container is bigger than 65535
`ERROR_PROFTRANS_MULTIPLE_SHOTS_ACTIV`	-111	A MultiShot request is active -> call `MultiShot(0)` for abort

2.11.4. Get profile

```
int GetProfile();
```

Transfer of a profiles by a serial interface. This function is only available for the serial interface.

2.11.5. Get actual profile

```
int GetActualProfile(unsigned char *pBuffer, unsigned int nBuffersize,
                 TProfileConfig ProfileConfig, unsigned int *pLostProfiles)
```

Fetching the active profile / container / video image. For it the function has to be passed a buffer in which the profile, the container and the video image is copied. By the parameter `ProfileConfig` the profile configuration requested for this interrogation may be selected (see chapter 2.21 “Profile / container / video transfer” and chapter 2.9.5 “Profile config”). If the container mode is activated the container may only be fetched with `PROFILE`.
By the pointer `pLostProfiles` the count of lost profiles / container may be interrogated. This value is higher than 0 if between two interrogations of the function several profiles / container have been received. If this parameter is not necessary alternatively also `NULL` may be passed.

The return value is the count of bytes copied into the buffer. If the count is less than or equal to `GENERAL_FUNCTION_NOT_AVAILABLE`, it is one of the general or following return values (see chapter 2.2 “General return values of the functions”):

Constant for the return value	Value	Description
`ERROR_PROFTRANS_WRONG _PROFILE_CONFIG`	-102	Not able to convert the loaded profile into the requested profile configuration
`ERROR_PROFTRANS_FILE_EOF`	-103	The end-of-file during the loading of profiles has been reached
`ERROR_PROFTRANS_NO_NEW _PROFILE`	-104	Since the last call of `GetActualProfile` no new profile has been received
`ERROR_PROFTRANS_BUFFER _SIZE_TOO_LOW`	-105	The buffer size of the passed buffer is too small
`ERROR_PROFTRANS_NO_PROFILE _TRANSFER`	-106	The profile transfer has not been started and no file is loaded

More details concerning profile transfer in chapter 2.21 “Profile / container / video transfer”.

2.11.6. Trigger Profile

```
int TriggerProfile();
```

Pretend an external Trigger to the scanCONTROL. The trigger inputs should not connect and the scanCONTROL is configured for external trigger.

2.11.7. Conversion of profile data

```
int ConvertProfile2Values(const unsigned char *pProfile,
    unsigned int nResolution, TProfileConfig ProfileConfig,
    TScannerType ScannerType, unsigned int nReflection, int bConvertToMM,
    unsigned short *pWidth, unsigned short *pMaximum,
    unsigned short *pThreshold, double *pX, double *pZ,
    unsigned int *pM0, unsigned int *pM1)

int ConvertPartProfile2Values(const unsigned char *pProfile,
    TPartialProfile *pPartialProfile, TScannerType ScannerType,
    unsigned int nReflection, int bConvertToMM,
    unsigned short *pWidth, unsigned short *pMaximum,
    unsigned short *pThreshold, double *pX, double *pZ,
    unsigned int *pM0, unsigned int *pM1);
```

Conversion of profiles or partial profiles in millimetre values for the position and distance co-ordinate. Furthermore all other contained information may be extracted.
There are two versions of this function, one for default profiles and one for the `PARTIAL_PROFILE` format.

Parameter	**Description**
`const unsigned char *pProfile`	Pointer to the profile
`unsigned int nResolution`	Points per profile (64 ... 1024)
`TProfileConfig ProfileConfig`	Profile configuration (`PURE_PROFILE` ... `PROFILE`)
`TScannerType ScannerType`	Measurement range of the scanCONTROL (`LLT25` or `LLT100`)
`unsigned int nReflection`	Stripe to be read out (0 to 3)
`int bConvertToMM`	Conversion of the positions and distance co-ordinates in millimetre (0 = deactivated, 1 = activated)
`TPartialProfile *pPartialProfile`	Partial-profile
`unsigned short *pWidth`	Pointer to an array for the reflection widths
`unsigned short *pMaximum`	Pointer to an array for the maximum intensities
`unsigned short *pThreshold`	Pointer to an array for the Thresholds
`double *pX`	Pointer to an array for the position co-ordinates
`double *pZ`	Pointer to an array for the distance co-ordinates
`unsigned int *pM0`	Pointer to an array for the M0
`unsigned int *pM1`	Pointer to an array for the M1

The arrays must have at least the size of the resolution (points per profile) respectively the `PointCounts` at `PARTIAL_PROFILE`.
If not all information are necessary the information may also be passed a `NULL` instead of the array.

The functions are automatically performing a test which information are available in the respective profiles.

If the return value is bigger than 0 the function has been successful and in the single bits of the return value the valid arrays are coded.

Set Bit	Constant for the Bit	Description
8	`CONVERT_WIDTH`	The array for the reflection width has been filled with data
9	`CONVERT_MAXIMUM`	The array for the maximum intensity has been filled with data
10	`CONVERT_THRESHOLD`	The array for the threshold has been filled with data
11	`CONVERT_X`	The array for the position co-ordinates has been filled with data
12	`CONVERT_Z`	The array for the distance co-ordinates has been filled with data
13	`CONVERT_M0`	The array for the M0 has been filled with data
14	`CONVERT_M1`	The array for the M1 has been filled with data

If the count is less than or equal to `GENERAL_FUNCTION_NOT_AVAILABLE` it is one of the general or following return values (see chapter 2.2 “General return values of the functions”):

Constant for the return value	Value	Description
`ERROR_PROFTRANS_REFLECTION_NUMBER_TOO_HIGH`	-110	The count of the requested stripes is bigger than 3

For further information see chapter 2.21 “Profile / container / video transfer”.

2.12. Is functions

Functions for interrogate states and connections.

```
int IsInterfaceType (int iInterfaceType)
```

Is the scanCONTROL connected via a serial interface of the type “iInterfaceType”? Valid interface types are:

Constants for iInterfaceType	Value	Description
`INTF_TYPE_SERIAL`	1	A connection via the serial interface
`INTF_TYPE_ETHERNET`	3	An Ethernet connection via Ethernet

```
int IsTransferingProfiles()
```

Is the scanCONTROL currently transferring data?

The return value may be one of the following return values:

Constant for the return value	Value	Description
`IS_FUNC_YES`	1	Interrogated state or connection is active
`IS_FUNC_NO`	0	Interrogated state or connection is not active

2.13. Partial profile functions

The scanCONTROL offers the possibility to restrict the transferred profile. The advantage of this procedure is the lower size of the transferred data. Furthermore the not needed ranges of a profile may be cut out already directly in the scanCONTROL.
This feature is not available for the scanCONTROL 28xx until from DSP firmware version 13 and cannot be combined with the container mode.

Samples for the transfer of profiles with the “Partial Profile” are to be found in chapter 3 “LLT2800Samples”.

2.13.1. GetPartialProfileUnitSize

```
int GetPartialProfileUnitSize(unsigned int *pUnitSizePoint,
                              unsigned int *pUnitSizePointData)
```

This function returns the increments for adjusting the partial profile. If the parameter `pUnitSizePoint` is equal to the resolution of the unrestrained profile this feature is not supported yet by the firmware version of your scanCONTROL.

2.13.2. Get/SetPartialProfile

```
int GetPartialProfile(TPartialProfile *pPartialProfile)
int SetPartialProfile(TPartialProfile *pPartialProfile)
```

By this function the partial profile transfer of the scanCONTROL may be adjusted. Before the profile configuration has to be set to `PARTIAL_PROFILE` (see chapter 2.9.5 “Profile config”).

Parameter	**Meaning**
`nStartPoint`	Number of the first point to be transferred
`nStartPointData`	First byte of the points
`nPointCount`	Count of the points to be transferred
`nPointDataWidth`	Count of the bytes per point

All parameter of the `SetPartialProfile` function always have to be a multiple of the respective `nUnitSize` of the function `GetPartialProfileUnitSize`.

The return value may be one of the general or following return values (see chapter 2.2 “General return values of the functions”):

Constant for the return value	**Value**	**Description**
`ERROR_PARTPROFILE_NO_PART _PROF`	-350	The profile configuration is not set to `PARTIAL_PROFILE` -> call `SetProfileConfig(PARTIAL_PROFILE);`
`ERROR_PARTPROFILE_TOO_MUCH _BYTES`	-351	The count of bytes per point is too high -> change `nStartPointData` or `nPointDataWidth`
`ERROR_PARTPROFILE_TOO_MUCH _POINTS`	-352	The count of points is too high -> change `nStartPoint` or `nPointCount`
`ERROR_PARTPROFILE_NO_POINT _COUNT`	-353	`nPointCount` or `nPointDataWidth` is `0`
`ERROR_PARTPROFILE_NOT_MOD _UNITSIZE_POINT`	-354	`nStartPoint` or `nPointCount` are not a

		multiple of `nUnitSizePoint` (see chapter GetPartialProfileUnitSize 2.13.1 „GetPartialProfileUnitSize“)
`ERROR_PARTPROFILE_NOT_MOD_UNITSIZE_DATA`	-355	`nStartPointData` or `nPointDataWidth` are not a multiple of `nUnitSizePointData` (see chapter GetPartialProfileUnitSize 2.13.1 „GetPartialProfileUnitSize“)

The construction of a profile is described in chapter 2.22 “Description of the profile data”.
In the normal case only the position (x value) and the distance (z value) of the first stripe are required.

2.14. Time functions

```
void Timestamp2TimeAndCount(const unsigned char *pTimestamp,
                double *pTimeShutterOpen, double *pTimeShutterClose,
                unsigned int *pProfileCount)
```

This function evaluates the whole timestamp of a profile. It returns the timestamp of the beginning and the end of the shutter and the consecutive profile numbers.

```
void Timestamp2CmmTriggerAndInCounter(const unsigned char *pTimestamp,
                            unsigned int *pInCounter, bool *pCmmTrigger,
                            bool *pCmmActive, unsigned int *pCmmCount)
```

This function evaluates only the optional part of the timestamp of a profile. It returns the counter reading of the internal counter, the `CmmTrigger` and `CmmActive` flags as well as the CMM trigger counter.

The `CmmTrigger` flag is always set in the profile in which also a CMM trigger impulse has been put out. The `CmmActive` flag is always set when the CMM trigger is activated in general. For the not required parameter it may alternatively also be passed `NULL`.

```
#include <vector>

DWORD Resolution = 0;
double ShutterOpen, ShutterClose;
unsigned int ProfileCount, InCounter, CmmCount;
bool CmmTrigger, CmmActive;

//interrogation of the resolution
pInterfaceLLT->GetResolution(&Resolution);

//creation of a buffer for the profile
std::vector<unsigned char> ProfilData((Resolution * 4) + 16);

if(pInterfaceLLT->GetActualProfile
   (&ProfilData[0], ProfileData.size(), PURE_PROFILE, NULL) ==
  ProfileData.size())
{
  //decoding the timestamp and the profile counter
  pInterfaceLLT->Timestamp2TimeAndCount(&ProfilData[(Resolution * 4)],
        &ShutterOpen, &ShutterClose, ProfileCount);

  //decoding the optional counter
  pInterfaceLLT->Timestamp2CmmTriggerAndInCounter(
        &ProfilData[(Resolution * 4)], &InCounter, &CmmTrigger, &CmmActive,
        &CmmCount);
}
```

2.15. Post processing functions

By the post processing the scanCONTROL may apply several modules on the profiles. These modules are only available in special options of the scanCONTROL. Please see the respective documentation for more detailed information.

2.15.1. Read/Write post processing parameter

```
int ReadPostProcessingParameter(DWORD *pParameter, unsigned int nSize)
int WritePostProcessingParameter(DWORD *pParameter, unsigned int nSize)
```

Reading or Writing of the active post processing parameter. These parameters may be up to 1024 DWORD maximum.

The return value may be one of the general return values (see chapter 2.2 “General return values of the functions”).
The post processing may be activated by the „Processing profile data“ property (for this see chapter 2.8.7 “Processing profile data”).

2.15.2. Post processing results

Some of the post processing modules may integrate results from their calculations in the transferred profile. They are written on the position and distance co-ordinates of a reflection. The following function extracts the calculation results from a given profile:

```
int ConvertProfile2ModuleResult(const unsigned char *pProfileBuffer,
        unsigned int nProfileBufferSize,
        unsigned char *pModuleResultBuffer, unsigned int nResultBufferSize,
        TPartialProfile *pPartialProfile)
```

The function is passed the profile buffer, its size, a buffer in which the result is copied and whose size. If a profile is loaded from a file another pointer has to be passed to the active `TPartialProfile` structure.

The return value is the count of bytes copied into the buffer. If the count is less than or equal to `GENERAL_FUNCTION_NOT_AVAILABLE`, it is one of the general or following return values (see chapter 2.2 “General return values of the functions”):

Constant for the return value	Value	Description
ERROR_POSTPROCESSING_NO_PROF_BUFFER	-200	No profile buffer has been passed
ERROR_POSTPROCESSING_MOD_4	-201	The parameter `nStartPointData` or `nPointDataWidth` is not divisible by 4
ERROR_POSTPROCESSING_NO_RESULT	-202	No result block could be found in the profile
ERROR_POSTPROCESSING_LOW_BUFFERSIZE	-203	The buffer size for the result is too small
ERROR_POSTPROCESSING_WRONG_RESULT_SIZE	-204	The size of the result block in the profile is not correct

2.16. File functions

Functions for loading and saving profiles or profile streams. Profiles can be saved and loaded with the `AVI`-file format (a standardized video-format).

Samples for loading and saving of profiles are to be found in chapter 3 "LLT2800Samples".

2.16.1. Saving of profiles

```
int SaveProfiles(const char *pFilename, TFileType FileType)
```

Saving of profiles. The profiles thereby are saved with the active profile configuration. The file name has to be indicated including extension.
The file format can be chosen with the parameter `Filetype`.

File type	Value	Description
`AVI`	0	AVI-file
`CSV`	2	CSV-file (only for profiles)
`BMP`	3	BMP-file (only for video pictures)
`CSV_NEG`	4	CSV file (only for profiles) with reflected Z-axis

It is recommended to use the `AVI`-file format, this format is supported by the scanCONTROL software from Micro-Epsilon.

For finishing the saving `SaveProfiles(NULL,0)` has to be called. If the maximum file size is reached during saving an error message with the `ERROR_STOPSAVING` value is send (see chapter 2.10.2 "Message").

The return value may be one of the general or following return values (see chapter 2.2 "General return values of the functions"):

Constant for the return value	Value	Description
`ERROR_LOADSAVE_WRITING_LAST _BUFFER`	-50	Error in the deactivation of the saving the last profile of the file could be damaged or not all have been saved
`ERROR_LOADSAVE_AVI_NOT _SUPPORTED`	-58	The operating system don't support the AVI-format, please use windows 2000 or better
`ERROR_LOADSAVE_WRONG_PROFILE _CONFIG`	-60	The profile configuration or the file type mismatch to the transferred profiles/containers/video-pictures
`ERROR_LOADSAVE_NOT _TRANSFERING`	-61	The profile transfer is not active

2.16.2. Loading of profiles

```
int LoadProfiles(const char *pFilename, TPartialProfile *pPartialProfile,
        TProfileConfig *pProfileConfig, TScannerType *pScannerType,
        DWORD *pRearrengementProfile)
```

Loading of profiles from a file. Thereby five pointers have to be passed on variables for the file name, the partial profile configuration, the profile configuration, the scanCONTROL type (measuring range) and the `Rearrengement` property. The pointers for the profile configuration and the scanCONTROL type (measuring range) may be `NULL`.
`*.AVI` files can be loaded, which are saved with the LLT.dll, the LLT2800Demo program or the scanCONTROL programs from Micro-Epsilon.

After the loading the single profile in the file may be read out by the function `GetActualProfile` (see chapter 2.11.5 “Get actual profile”).
Thereby it has to be seen, that from the `PartialProfile` parameter the size for the buffer for a profile / container may be calculated (`PointCount` * `PointDataWidth`).

The return value is the count of profiles / containers in the loaded file. If the count is less than `0` it is one of the general or following return (see chapter 2.2 “General return values of the functions”):

Constant for the return value	Value	Description
`ERROR_LOADSAVE_WHILE_SAVE_PROFILE`	-51	File cannot be loaded, as saving is active
`ERROR_LOADSAVE_NO_PROFILELENGTH_POINTER`	-52	No pointer for the profile length has been passed
`ERROR_LOADSAVE_NO_LOAD_PROFILE`	-53	The filename is `NULL`, but no file is currently loaded
`ERROR_LOADSAVE_STOP_ALREADY_LOAD`	-54	On file has already been loaded, the loading has been stopped
`ERROR_LOADSAVE_CANT_OPEN_FILE`	-55	Cannot open file
`ERROR_LOADSAVE_INVALID_FILE_HEADER`	-56	The file header of the file to be loaded is wrong
`ERROR_LOADSAVE_AVI_NOT_SUPPORTED`	-58	The operating system don’t support the AVI-format, please use windows 2000 or better
`ERROR_LOADSAVE_NO_REARRANGEMENT_POINTER`	-59	The pointer `pRearrengementProfile` is `NULL`

The profile configuration of the loaded profile should always correspond to the profile configuration of the `LoadProfiles` function. Additionally at saved profile configuration of `PROFILE` also a `QUARTER_PROFILE` and `PURE_PROFILE` or at `QUARTER_PROFILE` also a `PURE_PROFILE` may be read out.

The loading of a file does not influence the profile configuration value for the saving of profiles respectively the callback.

For terminating the loading `LoadProfiles(NULL, NULL, NULL, NULL, NULL)` has to be called.

With the function `LoadProfilesGetPos` (see next chapter) the current count of profiles in the file may be interrogated.

2.16.3. Navigation of a loaded file

```
int LoadProfilesGetPos(unsigned int *pActualPosition,
                       unsigned int *pMaxPosition)
```

By this function the count of profiles and the active reading position in the loaded file may be interrogated. If in the file is still loaded by another instance the count of profiles can change.

```
int LoadProfilesSetPos(unsigned int nNewPosition)
```

Set up of the active reading position in the loaded file. For setting the position on the first profile `nNewPosition` has to be 0.

The return value may be one of the general or following return values (see chapter 2.2 “General return values of the functions”):

Constant for the return value	Value	Description
`ERROR_LOADSAVE_FILE_POSITION_TOO_HIGH`	-57	The requested position is bigger than or equal to the maximum position

2.17. Special CMM trigger functions

By the special CMM trigger functions the starting and terminating of the profile transfer with activated CMM trigger is simplified. Additionally the profiles with active CMM trigger may be saved in a file. The CMM trigger is only available for certain options of the scanCONTROL.
Concerning the CMM trigger see also the description in chapter 2.8.12 “CMM trigger”.

Samples for the CMM-Trigger are to be found in chapter 3 “LLT2800Samples”.

Starting the profile transfer with CMM trigger:

```
int StartTransmissionAndCmmTrigger(DWORD cmmTrigger,
                    unsigned int nProfilesForerun, const char *pFilename,
                    TFileType FileType, unsigned int nTimeout);
```

Description of the parameter:

Parameter	Description
`nCmmTrigger`	First indicator word of the CMM trigger, which contains the divisor and the polarity
`nProfilesForerun`	Count of the continuous received profiles from which a stable data transfer is adopted
`pFilename`	File name for the file to be saved (has to be `NULL` if no saving is wanted)
`FileType`	File format of the saved file (see chapter 2.16.1 „aving of profiles“)
`nTimeout`	Timeout in ms for the whole function

The return value may be one of the general or following return values (see chapter 2.2 “General return values of the functions”):

Constant for the return value	Value	Description
`ERROR_CMMTRIGGER_NO_DIVISOR`	-400	Divisor has to be > 0
`ERROR_CMMTRIGGER_TIMEOUT _AFTER_TRANSFERPROFILES`	-401	After `TransferProfiles` no profiles have been received
`ERROR_CMMTRIGGER_TIMEOUT _AFTER_SETCMMTRIGGER`	-402	After setting of the CMM trigger not enough profiles with active CMM trigger have been received

For using this function you have to set the second till fourth instruction word of the CMM triggers before calling this function. The first instruction word is only set by this function.

The `StartTransmissionAndCmmTrigger` function starts first profile transfer without forwarding the profiles thereby by callback. If the connection has run in, i.e. the requested count of profiles has been transferred without failure, the first CMM trigger command with the divisor is send to the scanCONTROL. Afterwards it is waiting for the first profile with active CMM trigger flag. From this profile on all further profiles are forwarded by callback. Additionally, if a file name has been passed, the saving of the profiles with the passed file names is started.
If a timeout occurs during the waiting for the profile the function will be aborted.

It is reasonable to indicate for `nProfilesForerun` half a profile rate (e.g. 500 profiles at 1000 Hz) and for the `nTimeout` 3000 ms.

Stopping the profile transfer with CMM trigger:

```
int StopTransmissionAndCmmTrigger(int nCmmTriggerPolarity,
                                  unsigned int nTimeout)
```

Description of the parameter:

Parameter	**Description**
nCmmTriggerPolarity	Polarity of the CMM triggers (0 = Low active, 1 = High active)
nTimeout	Timeout in ms for the whole function

The return value may be one of the general or following return values (see chapter 2.2 “General return values of the functions”):

Constant for the return value	**Value**	**Description**
`ERROR_CMMTRIGGER_TIMEOUT_AFTER_SETCMMTRIGGER`	-402	After setting the CMM trigger no profile with deactivated CMM trigger has been received

The `StopTransmissionAndCmmTrigger` function first stops the CMM trigger, by setting the divisor to 0 (but thereby taking into account the passed polarity). Afterwards it is waiting for the first profile without active CMM trigger flag. This profile and all the following will not be forwarded by callback, the profile transfer will be stopped and in case of saving the saving will be terminated.
If a timeout occurs during waiting for the first profile without active CMM trigger flag the function will be aborted.
It is reasonable to indicate a time between 100 and 500 ms for the `nTimeout`.

After stopping the transfer the second till fourth CMM-Trigger-instruction words remain preserved and have not to be reset.

2.18. Error value conversion functions

Conversion of an error value in an error text.

```
int TranslateErrorValue(int ErrorValue, char *pString,
                        unsigned int nStringSize);
```

This function is passed the error value to be converted and a buffer for the string.

The return value is the count of characters copied into the buffer. If the count is less than or equal to `GENERAL_FUNCTION_NOT_AVAILABLE`, it is one of the general or following return values (see chapter 2.2 "General return values of the functions"):

Constant for the return value	Value	Description
ERROR_TRANSERRORVALUE_WRONG _ERROR_VALUE	-450	A wrong error value has been passed
ERROR_TRANSERRORVALUE_BUFFER _SIZE_TO_LOW	-451	The size of the passed buffer is too small for the string

```
char ErrorString[200];

//conversion of an error value into a string
if(pInterfaceLLT->TranslateErrorValue(GENERAL_FUNCTION_NOT_AVAILABLE,
        ErrorString, sizeof(ErrorString)) > GENERAL_FUNCTION_NOT_AVAILABLE)
{
  //if the conversion has been successful -> output of the string
  cout << ErrorString;
}
```

2.19. ExportLLTConfig

```
int ExportLLTConfig(const char *pFileName)
```

Exporting the current configuration of the scanCONTROL. This configuration file contains all relevant parameters and is primarily intended for post processing applications.
The file format complies with the communications protocol for the serial connection with the scanCONTROL. The configuration files thus created can be transmitted to the scanCONTROL without changes via the serial port using a terminal program.

Constants for the return value	Value	Description
ERROR_READWRITECONFIG_CANT _CREATE_FILE	-500	The specified file cannot be created.

```
int ExportLLTConfigString(char* configData, int configDataSize)
```

Exporting the current configuration of the scanCONTROL. If the data pointer is 0, the return value is the required buffer size. Otherwise it is one of the general return values (see chapter 2.2 "General return values of the functions").

2.20. ImportLLTConfig

```
int ImportLLTConfig(const char* pFileName)
int ImportLLTConfigString(const char* configData, int configDataSize)
```

Import configurations, which create with the equivalent export function (see section 2.19.).

2.21. Profile / container / video transfer

The profiles / container / video pictures are regularly fetched from the driver by the LLT.dll.
If the scanCONTROL is connected by a serial interface only `PURE_PROFILE` profiles may be processed. This restriction results from the limited speed of the serial interface.
If the scanCONTROL on the other hand is connected by ethernet the active profile configuration for the saving or the callback may be adjusted by the function `SetProfileConfig` (see chapter 2.9.5 "Profile config").

The callback is always called when a new profile or a new container has been received. The callback can't be use for video-pictures. Thus it serves as notification. As parameter this callback contains a pointer on the just received profile / container. But the profile has before if necessary already been changed into the active profile configuration. By this pointer the callback function may copy the profile or container in a suitable buffer for further processing. Thereby it is important that the callback function is very short and terminated as soon as possible before the next buffer may be fetched from the driver.

For applications less time-critical respectively applications, which do not have to process all profiles the function `GetActualProfile` is intended (see chapter 2.11.5 "Get actual profile"). This function has to be passed a pointer to a buffer in which the active profile, container or video picture may be copied by the LLT.dll. thereby also the requested profile configuration has to be indicated, which is independent from the profile configuration for the saving and the callback.
By this function also profile may be read in from a file.

By the following table the profile size the callback is passing or which size the field for the `GetActualProfile` function has to be may be calculated.

ProfileConfig	**Description**	**Profile byte**
`PROFILE`	Profile data of all four stripes	64 * resolution
`QUARTER_PROFILE`	Profile data of one stripe	16 * resolution + 16
`PURE_PROFILE`	Reduced profile data of one stripe (only the position and distance values)	4 * resolution + 16
`PARTIAL_PROFILE`	Partial profile	`PointCount` * `PointDataWidth` of the `TPartialProfile` structure
`CONTAINER`	Container data	Height * width of the container
`VIDEO_IMAGE`	Video image of the scanCONTROL	Height * width of the video image

```
#include <vector>
//Setting of the resolution
pInterfaceLLT->SetResolution(256);

//activation of the profile transfer and waiting one moment (for profile)
pInterfaceLLT->TransferProfiles(NORMAL_TRANSFER, true);
Sleep(500);

//creation of a buffer for the profile in QUARTER_PROFILE Mode and
//interrogation of a profile
std::vector<unsigned char> vProfile(256*16+16);
if(pInterfaceLLT->GetActualProfile(&vProfile[0], ProfSize, QUARTER_PROFILE,
                                   NULL) != 256*16+16)
{
   //the scanCONTROL does not send profiles
  return;
}

double XValue[256], ZValue[256];

//conversion of the x and z co-ordinates of the profile in millimetres
int RetValue = pInterfaceLLT->ConvertProfile2Values(&vProfile[0], 256,
   QUARTER_PROFILE, 0, 1, NULL, NULL, NULL, &X[0], &Z[0], NULL, NULL);

if((RetValue & CONVERT_X > 0) && (RetValue & CONVERT_Z > 0))
{
  //the x and z co-ordinates have been converted in millimetre
}
```

2.22. Description of the profile data

The profile data have a width of 64 Byte. The height of the profile data is equal to the amount of points per profile.
The profile data consists of 4 stripes to each 16 Byte:

Stripe 1	Stripe 2	Stripe 3	Stripe 4

Each stripe contains the data for one profile. By default the 1st stripe contains the measured profile and the further stripes invalid data (except for the timestamp at the end of the 4th stripe). All stripes in special cases may contain valid profiles (e.g. if all stripes are linearised). Furthermore the results for the post processing may be in the stripes 1 till 4.

Each line in a stripe contains the data for a point and has the following structure:

Res. (2 Bit)	Width (10 Bit)	Height (10 Bit)	Threshold (10 Bit)

Position (16 Bit)	Distance (16 Bit)

Moment 0 (32 Bit)

Moment 1 (32 Bit)

The single data of a point are described in the following table:

Amount of the Bits	Name	Description
2	Res.	Reserved
10	Width	Width of the reflection in Pixel
10	Height	Maximum intensity of the reflection above the threshold
10	Threshold	Actual threshold
16	Position	Position co-ordinate (X)
16	Distance	Distance co-ordinate (Z)
32	Moment 0	Integral intensity of the reflection
32	Moment 1	1st moment

The data is provided in **Big-Endian**-format.

The position (X) and the distance (Z) are transferred as integer values and still have to be converted into millimetres (see 2.11.7 “Conversion of profile data”).
If the position or the distance is 0 before the conversion, the point is invalid, this means that scanCONTROL could not define a distance for this point.

In the normal case only the 1st stripe is used. Only in very special cases it is reasonable to process all stripes.

In the last point of the 4th stripe the timestamp is situated.

Samples for the transfer of profiles with different profile configurations and the conversion of profile data in millimetre are to be found in chapter 3 “LLT2800Samples”.

2.22.1. Description of the data format PROFILE

This predefined format is transferred by scanCONTROL by default. It consists of 4 stripes.
The size of this format amounts to: 64 * resolution Bytes. This data format uses the **Big-Endian**-format.

2.22.2. Description of the data format QUARTER_PROFILE

This predefined format may generate the LLT.dll for data reduction from the `PROFILE` format. It only consists of one stripe. The timestamp is situated in the 16 Bytes after the last point. The size of this format amounts to: 16 * resolution + 16 Bytes. This data format uses the **Big Endian** format.

2.22.3. Description of the data format PURE_PROFILE

This predefined format may generate the LLT.dll for data reduction from the `PROFILE` format. It only consists of the position and distance values of the respectively selected stripe. They are forwarded successively as WORDs (2 Bytes). The timestamp is situated in the 16 Bytes after the last point.

The size of this format amounts to: 4 * resolution + 16 Bytes

This data format uses the **Little Endian** format.

2.22.4. Description of the data format PARTIAL_PROFILE

The `PARTIAL_PROFILE` is directly generated in the scanCONTROL. The size and the significance of the data of the profile transferred herewith, depends on the settings of the function `SetPartialProfile` (see chapter 2.13 "PartialProfile functions" and 2.22 "Description of the profile data").
The timestamp in this format is always situated in the last 16 Bytes.

In the following table the settings of the "Partial Profile" for the profile configurations `QUARTER_PROFILE` and `PURE_PROFILE` are listed.

ProfileConfig	**StartPoint**	**StartPointData**	**PointCount**	**PointDataWidth**
`QUARTER_PROFILE`	0	0	Resolution	16
`PURE_PROFILE`	0	4	Resolution	4

This data format uses the **Big-Endian** format.

2.23. Description of the data format CONTAINER

In the container mode several profiles are combined to a container / image. In the columns the single points with the selected properties and in the lines the single profiles are listed.

For example

	Z Point 1	**...**	**Z point n**	**X point 1**	**...**	**X point n**
Profile 1						
Profile 2						
Profile 3						
Profile 4						

The resolution of one point is 16 bit. This container may be shown directly as 16 bit grey scale bitmap or characters may be extracted by image processing algorithms.
In the last 16 byte of each line the timestamp of the profiles may be shown.
The width of the image is defined by the settings of the `Rearrangement` property (see chapter 2.8.13 "Rearrangement profile"). The height (= the count of profiles per container) may be free chosen (see chapter 2.9.7 "Profile container size").

This data format uses the **Big Endian** format.

2.24. Description of the data format VIDEO_IMAGE

The video-images are transferred as 8 bit greyscale bitmap. With it only the pure image data is transferred. Eventual header or the reflection of the lines has to be realized external. This data format has no timestamp.

2.25. Description of the timestamp

The timestamp is situated in the last 16 bytes of a profile. The time of the timestamp starts every 128 seconds again at 0.
The timestamp consists of the times of the start and the end of the shutter of a profile and a consecutive profile number.

The timestamp uses the **Big Endian** format.

Flags (2 bit)	Reserved (6 bit)	Profile counter (24 bit)

Timestamp of the shutter time start (32 bit)

0x00000000 (32 bit)

Timestamp of the shutter time end (32 bit)

If the CMM-trigger (an optional trigger) or the internal counter (an optional counter) is used, the 3rd sector of the timestamp has the following significance:

Flags (2 bit)	Reserved (6 bit)	Profile counter (24 bit)

Timestamp of the shutter time start (32 bit)

Flank counter_2 (16 bit)	CMM trigger flag (1 bit)	CMM active flag (1 bit)	CMM trigger pulse counter (bit 14)

Timestamp of the shutter time end (32 bit)

The 32 bit of the timestamp are composed as follows:

Bit Position	Description
31..25	Seconds
24..12	Cycle (overrun at 8000)
11..00	Cycle offset (overrun at 3072)

For simplification a conversion function has been integrated in the DLL which ex-decodes the single timestamps and counter (see chapter 2.15 “Time functions”).

3. LLT2800Samples

The sample programs in the LLT2800Samples directory are intended as examples for the integration of the scanCONTROL in own projects. For a first view it is delivered complete with source program.

Directory	**Description**
LLTInfo	Simple connect to the scanCONTROL with interrogation of name and serial number
GetProfiles_Poll	Transfer of profiles to the LLT.dll and pick up of profiles in polling mode by a sample program
GetProfiles_Ethernet	Transfer of profiles to the LLT.dll and pick up of profiles in polling mode by a sample program over Ethernet
GetProfiles_Callback	Transfer of profiles to the LLT.dll and pick up of profiles with a callback by a sample program
GetProfiles_Serial	Transmission of profiles to the LLT.dll and reading the profiles via the serial port
MultiShot	Transfer of a defined count of profiles from the scanCONTROL
PartialProfile	Transfer of a partial profile
LoadSave	Loading and saving of profiles
ContainerMode	Transfer of profile-containers respectively gray-scale maps
VideoMode	Transfer of video-pictures from the sensor-matrix
MultiLLTs	Usage of more than one scanCONTROL in one program
CmmTrigger	Usage of the optional programmable trigger (CMM trigger)
CSharp_GetProfiles_Poll	Transfer of profiles to the LLT.dll and pick up of profiles in polling mode by a sample program in C#
bin	Compiled sample programs for testing