LLT.DLL

Schnittstellendokumentation

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

Schnittstellendokumentation zur LLT.dll

Die LLT.dll ist eine DLL zum einfachen Integrieren des scanCONTROL's in eigene Anwendungen. Sie bildet eine Abstraktionsebene über dem direkten Ansprechen des scanCONTROL per Ethernet oder der seriellen Schnittstelle. Beim Design dieser DLL wurde besonderer Wert auf die Einfachheit der Schnittstelle und eine hohe Performance gelegt.
Um diese DLL mit möglichst vielen verschiedene Entwicklungsumgebungen und Compilern nutzen zu können, wurde die DLL Schnittstelle mit reinen C-Funktionen der „cdecl" und der „stdcall" Aufrufkonvention realisiert. Dadurch kann die DLL auch unter C, Delphi oder anderen Programmiersprachen genutzt werden (Bedingung dafür ist die Kompatibilität der verwendeten Datentypen). Für C++ Anwendungen gibt es eine zusätzliche Klasse mit deren Hilfe die C-Funktionen in Methoden einer Interface-Klasse gemappt werden.

In dieser Dokumentation wird nur die Einbindung der DLL in C++ beschrieben, die Einbindung in C oder in andere Programmiersprachen kann aus dieser Dokumentation abgeleitet werden.

Bei scanCONTROL, welche über Ethernet verbunden sind, ist es für Debugging-Zwecke nötig das Heartbeat Timeout zu erhöhen (siehe Kapitel 2.9.10 Ethernet Heartbeat timeout). Ansonsten kann es während des Debuggens zu Verbindungsabbrüchen zum scanCONTROL kommen.

1. Laden der DLL

Für das Laden einer DLL gibt es zwei verschiedene Möglichkeiten. Sie kann direkt beim Starten der Applikation geladen werden (statisch) oder später bei bedarf dynamisch. Das dynamische Laden ist meist günstiger, vor allem weil es eine bessere Fehlerbehandlung ermöglicht.

1.1. Statisches Laden

Beim statischen Laden der DLL in ein C oder C++ Projekt, wird die dazugehörige *.lib-Datei mit den Definitionen der DLL-Funktionen in das Projekt compiliert. Dabei ist es wichtig, dass immer die zu der verwendeten DLL-Version passende .lib-Datei verwendet wird. In den Header-Dateien `C_InterfaceLLT_2.h` und `S_InterfaceLLT_2.h` sind die zugehörigen Funktions-Deklarationen zu finden. Dabei steht der Präfix `s_` für „stdcall" und `c_` für „cdecl".

Für andere Programmiersprachen können anhand der `C_InterfaceLLT_2.h` oder der `S_InterfaceLLT_2.h` Importfunktionen für die DLL entwickelt werden.

Bei einer neuen DLL-Version muss das Projekt neu übersetzt werden, da sich die Einspringpunkte in der DLL ändern können.

1.2. Dynamisches Laden

Zum Laden der LLT.dll und Importieren ihrer Funktionen in C++-Projekte werden die zwei Klassen `CInterfaceLLT` und `CDllLoader` bereitgestellt.
Der `DllLoader` ist für das DLL handling (laden und Abfragen der Funktionspointer) zuständig. In der Interface-Klasse (`CInterfaceLLT`) sind alle Funktionen die die LLT.dll

exportiert als Funktionspointer definiert. Sie können deshalb einfach als Methoden der

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

...

//Anlegen der Interface-Klasse
pInterfaceLLT = new CInterfaceLLT();

//Testen ob es die gewuenschte Funktion gibt
if(pInterfaceLLT->m_pFunctions->CreateLLTDevice!= NULL)
{
  //Aufrufen von Funktionen in der LLT.dll
  pInterfaceLLT->CreateLLTDevice(INTF_TYPE_ETHERNET);
}
```

Interface-Klasse aufgerufen werden.

Der herausragendste Vorteil dieser Interface-Klasse ist das dynamische Abfragen der Funktionen der DLL. Das heißt, es kann ohne das Projekt neu zu übersetzen eine neue DLL-Version eingesetzt werden. Zusätzlich kann noch abgefragt werden, ob die gewünschte Funktion in der DLL vorhanden ist. Dies ist aber nur für Funktionen, die nach dem ersten Release hinzugekommen sind, notwendig.
Sollen neue Funktionen der LLT.dll verwendet werden, muss das Projekt mit der aktuellen Interface-Klasse neu übersetzt werden.

Zusätzlich kann dem Konstruktor der Interface-Klasse noch der Name der zu ladenden DLL (mit Pfad) und ein Pointer auf eine `bool`-Variable übergeben werden, welche einen Fehler beim Laden der DLL signalisiert.

2. Beschreibung der einzelnen Funktionen

Die Funktionen der LLT.dll gliedern sich in mehrere Funktionsgruppen:

Funktionsgruppe	Beschreibung
Instanz-Funktionen	Zum Erstellen einer scanCONTROL-Instanz mit Ethernet oder serieller Schnittstellenunterstützung und zum Löschen dieser Instanz
Auswahl-Funktionen	Zum Auswählen eines scanCONTROLs
Verbindungs-Funktionen	Verbinden und Schließen einer Verbindung mit einem Gerät
Identifikations-Funktionen	Abfragen des Namens und der Version
Eigenschafts-Funktionen	Abfragen und Setzen von Eigenschaften
Spezielle Eigenschafts-Funktionen	Abfragen und Setzen von speziellen Eigenschaften
Register Funktionen	Zum Registrieren von einem Callback und einer Error-Message
Profilübertragungs-Funktionen	Übertragen von Profilen
Is Funktionen	Zum Abfragen von verschiedenen Zuständen und Verbindungen
PartialProfile Funktionen	Einschränkung des zu übertragenden Profils auf dem scanCONTROL
Time Funktionen	Auswerten des Timestamps
Postprocessing Funktionen	Lesen und Schreiben der Postprocessing-Parameter
File Funktionen	Laden und Speichern von Profilen
Spezielle CMM-Trigger Funktionen	Starten und Stoppen der Profilübertragung inklusive des CMM-Triggers
Fehlerwert Konvertierungs-Funktion	Konvertieren des Fehlerwertes von Funktionen in Text

Für alle die hier beschriebenen Funktionen sind immer die Parameter für die `CInterfaceLLT`-Klasse angegeben. Wird diese Klasse nicht verwendet hat jede dieser Funktionen einen zusätzlichen ersten Parameter (`Instanzhandle`). Alle andere Parameter der Funktionen verschieben sich um eins nach hinten (siehe die Datei `C_InterfaceLLT_2.h` oder `S_InterfaceLLT_2.h`). Dieser zusätzliche erste Parameter dient zur Unterscheidung der Verschiedenen scanCONTROL-Instanzen in der DLL. In der `CInterfaceLLT`-Klasse wird dieser Parameter automatisch eingefügt.

2.1. Funktionen der LLT.dll bis Version 3.0.0

Um den neuen Verbindungstyp Ethernet zu unterstützen, wurden die neuen Instanz-Funktionen „CreateLLTDevice“, „IsInterfaceType“ und „GetInterfaceType“ hinzugefügt. Da diese nun einheitlich für alle Verbindungstypen genutzt werden sollen sind die bisherigen Funktionen als veraltet zu betrachten. Sie dienen lediglich der Abwärtskompatibilität der LLT.dll.

Alter Funktionsname	**Neuer Funktionsname**	**Kapitel**
`CreateLLTSerial`	`CreateLLTDevice(INTF_TYPE_SERIAL)`	2.3
`IsSerial`	`IsInterfaceType(INTF_TYPE_SERIAL)`	2.12

2.2. Allgemeine Rückgabewerte der Funktionen

Alle Funktionen des Interfaces geben einen `int` Wert als Rückgebewert zurück. Ist der Rückgabewert einer Funktion größer oder gleich `GENERAL_FUNCTION_OK` bzw. `'1'` so war die Funktion erfolgreich, ist der Rückgabewert `GENERAL_FUNCTION_NOT_AVAILABLE` bzw. `'0'` oder negativ so ist ein Fehler aufgetreten.
Eine Besonderheit gibt es bei einigen Funktionen, die auch `GENERAL_FUNCTION_CONTAINER_MODE_HEIGHT_CHANGED` bzw. `'2'` zurückgeben können. Tritt dieser Rückgabewert auf, hat sich die Größe des Bildes im Container-Mode geändert (siehe Kapitel 2.11 „Profilübertragungs-Funktionen“).

Zur Unterscheidung der einzelnen Rückgabewerte stehen mehrere Konstanten zu Verfügung. In der folgenden Tabelle sind alle allgemeinen Rückgabewerte aufgeführt, die von Funktionen zurückgegeben werden können. Für die einzelnen Funktionsgruppen kann es zusätzlich noch spezielle Rückgabewerte/Fehlerwerte geben.

Konstante für den Rückgabewert	Wert	Beschreibung
`GENERAL_FUNCTION_CONTAINER_MODE_HEIGHT_CHANGED`	2	Funktion erfolgreich ausgeführt, aber die Bild-Größe für den Container-Mode wurde verändert
`GENERAL_FUNCTION_OK`	1	Funktion erfolgreich ausgeführt
`GENERAL_FUNCTION_NOT_AVAILABLE`	0	Diese Funktion ist nicht verfügbar, ev. eine neue DLL verwenden oder in den Ethernet-Mode wechseln
`ERROR_GENERAL_WHILE_LOAD_PROFILE`	-1000	Funktion konnte nicht ausgeführt werden, da das Laden von Profilen aktiv ist
`ERROR_GENERAL_NOT_CONNECTED`	-1001	Es besteht keine Verbindung zum scanCONTROL -> `Connect` aufrufen
`ERROR_GENERAL_DEVICE_BUSY`	-1002	Die Verbindung zum scanCONTROL ist gestört oder getrennt -> neu Verbinden und Anschluss des scanCONTROL’s überprüfen
`ERROR_GENERAL_WHILE_LOAD_PROFILE_OR_GET_PROFILES`	-1003	Funktion konnte nicht ausgeführt werden, da entweder das Laden von Profilen oder die Profilübertragung aktiv ist
`ERROR_GENERAL_WHILE_GET_PROFILES`	-1004	Funktion konnte nicht ausgeführt werden, da die Profilübertragung aktiv ist
`ERROR_GENERAL_GET_SET_ADDRESS`	-1005	Die Adresse konnte nicht gelesen oder geschrieben werden. Eventuell wird eine zu alte Firmware verwendet.
`ERROR_GENERAL_POINTER_MISSING`	-1006	Ein benötigter Pointer ist `NULL`
`ERROR_GENERAL_WHILE_SAVE_PROFILES`	-1007	Funktion konnte nicht ausgeführt werden, da das Speichern von Profilen aktiv ist
`ERROR_GENERAL_SECOND_CONNECTION_TO_LLT`	-1008	Es ist eine zweite Instanz über Ethernet oder die serielle Schnittstelle mit diesem scanCONTROL verbunden. Bitte schließen Sie die zweite Instanz.

2.3. Instanz-Funktionen

Die LLT.dll unterstützt die Kommunikation zum scanCONTROL über die serielle Schnittstelle und über Ethernet. Nach dem Laden der DLL muss mit `CreateLLTDevice(InterfaceType)` ein entsprechendes Device in der DLL angelegt werden. Bei Erfolg geben diese Funktionen `GENERAL_FUNCTION_OK` zurück.
Der zur Erstellung eines Devices verwendete InterfaceType kann über die Funktion `GetIntefaceType()` abgefragt werden.

Unterstützt werden folgende InterfaceType Werte:

Konstante für InterfaceType	Wert	Beschreibung
`INTF_TYPE_UNKNOWN`	0	Wird von GetInterfaceType im Fehlerfall zurückgegeben, für CreateLLTDevice unzulässig.
`INTF_TYPE_SERIAL`	1	Eine Verbindung mittels der seriellen Schnittstelle
`INTF_TYPE_ETHERNET`	3	Eine Ethernetverbindung mittels Ethernet

Wird die `CInterfaceLLT`-Klasse nicht verwendet, geben diese beiden Funktionen statt `GENERAL_FUNCTION_OK` oder `GENERAL_FUNCTION_NOT_AVAILABLE` ein `Instanzhandle` für die in der LLT.dll angelegte interne Instanz zurück. Dieses Handle muss allen weiteren Funktionen als erster Parameter mit übergeben werden.
Ist dieses `Instanzhandle 0` oder `0xffffffff` ist das Erstellen eines Devices fehlgeschlagen.

Mit Hilfe der Funktion `DelDevice()` muss vor dem Entladen der DLL das angelegte Device wieder gelöscht werden (dies wird in der `CInterfaceLLT` Klasse automatisch gemacht).

Bei einem `DelDevice()` bleiben alle Parameter die im scanCONTROL eingestellt wurden erhalten, außer die „Profile config“ (siehe Kapitel 2.9.6), die „Packet size“ (siehe Kapitel 2.9.5) und der „Buffer count“ im Treiber (siehe Kapitel 2.9.2).

2.4. Parallelbetrieb mehrerer scanCONTROLs

Um mehrere scanCONTROL’s parallel in einem Programm zu nutzen gibt es zwei Möglichkeiten, abhängig von der gewählten Methode zum Laden der LLT.dll.

Statisches Laden:
Pro verbundenen scanCONTROL muss jeweils die Funktion `CreateLLTDevice()` aufgerufen werden. Durch die unterschiedlichen zurückgegebenen Instanzhandle können die verschiedenen scanCONTROL’s bzw. Instanzen unterschieden werden.

Dynamisches Laden:
Pro verbundenen scanCONTROL muss jeweils eine Instanz der CInterfaceLLT – Klasse angelegt werden. Die Instanzen der Klasse verwalten selbstständig den Instanzhandle für jedes scanCONTROL.

Sollen Callbacks verwendet werden können die unterschiedlichen Instanzen sich einen Callback teilen (siehe Kapitel 2.10.1 „Callback“).

```
#include <vector>

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

  //Erstellen von zwei Ethernet Devices/Instanzen
  pInterfaceLLT_1->CreateLLTDevice(INTF_TYPE_ETHERNET);
  pInterfaceLLT_2->CreateLLTDevice(INTF_TYPE_ETHERNET);

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

  //Auslesen der verfuegbaren Interfaces
  int InterfaceCount = pInterfaceLLT_1->GetDeviceInterfaces(
    & EthernetInterfaces[0], EthernetInterfaces.size());

  if(InterfaceCount < 2)
  {
    //Es ist nur 1 scanCONTROL verbunden
    return;
  }

  //Setzen der verschiedenen Interfaces
  pInterfaceLLT_1->SetDeviceInterface(EthernetInterfaces[0], 0);
  pInterfaceLLT_2->SetDeviceInterface(EthernetInterfaces[1], 0);

  //Verbinden der zwei scanCONTROL's
  pInterfaceLLT_1->Connect();
  pInterfaceLLT_2->Connect();

  //Registrieren des Callbacks
  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 Aufgerufen von LLT_1
    }
    case 1:
    {
      //Callback Aufgerufen von LLT_2
    }
  }
}
```

2.5. Auswahl-Funktionen

Mit Hilfe der Auswahlfunktionen kann eine Auswahl des Device-Interfaces erfolgen. Zum Beispiel kann damit die verwendete serielle Schnittstelle oder die IP-Adresse ausgewählt werden.

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

Abfrage der Device-Interfaces an die ein scanCONTROL angeschlossen ist. Der `Interfaces`-Parameter ist ein Feld aus Integer-Variablen, in das die Interface-Nummern eingetragen werden. `Size` gibt dabei die Größe des Feldes an.
Bei der Kommunikation über die serielle Schnittstelle werden die Nummern aller benutzbaren Com-Ports (Port 1 bis 16) und bei einer Verbindung über Ethernet die IP-Adressen aller gefundenen Geräte.
Der Rückgabewert gibt die Anzahl der gefundenen und in das `Interfaces`-Feld eingetragenen Device-Interfaces zurück.
Die Fast-Funktion liefert schneller eine Antwort bei einer Ethernet Verbindung, es kann aber sein das in großen Netzen nicht alle Sensoren gefunden werden.
Ist der Rückgabewert kleiner oder gleich `GENERAL_FUNCTION_NOT_AVAILABLE` kann er einer der allgemeinen oder der folgenden Rückgabewerte sein (siehe Kapitel 2.2 „Allgemeine Rückgabewerte der Funktionen“):

Konstante für den Rückgabewert	Wert	Beschreibung
`ERROR_GETDEVINTERFACES_WIN_NOT_SUPPORTED`	-250	Funktion steht nur unter Windows 2000 oder höher zur Verfügung
`ERROR_GETDEVINTERFACES_REQUEST_COUNT`	-251	Die Größe des übergebenen Feldes ist zu klein
`ERROR_GETDEVINTERFACES_INTERNAL`	-253	Bei der Abfrage der angeschlossenen scanCONTROL ist ein Fehler aufgetreten

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

Setzen der Nummer eines Interfaces und Übergabe eines zusätzlichen Parameters. Dieser Zusatzparameter wird bei einer Verbindung über die serielle Schnittstelle verwendet, um die Baudrate der seriellen Schnittstelle anzugeben, daher muss der Wert 115 200 betragen. Bei der Ethernet Schnittstelle entspricht nInterface der IP-Adresse in der sogenannten Host-Byte-Order, dabei werden die 4 Teile der IP-Adresse der Reihe noch von höchstem zum niedrigsten Byte gespeichert. Dies kann durch eine einfache Shift-Operation der einzelnen Bytes der IP-Adresse erreicht werden, hier ein Beispiel für die Adresse 192.168.1.2:

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

Der Zusatzparameter kann bei der Verwendung der Ethernet Schnittstelle verwendet werden um die lokale IP- Adresse des Rechners anzugeben, an die der Sensor die Daten schicken soll. Dies kann notwendig sein um Probleme bei Rechnern mit mehreren Netzwerkkarten zu lösen. Standardmäßig sollte jedoch 0 angegeben werden, dann bestimmt die LLT.DLL selbst die optimale IP- Adresse.

Ist der Rückgabewert kleiner oder gleich `GENERAL_FUNCTION_NOT_AVAILABLE` kann er einer der allgemeinen oder der folgenden Rückgabewerte sein (siehe Kapitel 2.2 „Allgemeine Rückgabewerte der Funktionen“):

Konstante für den Rückgabewert	Wert	Beschreibung
`ERROR_GETDEVINTERFACES_CONNECTED`	-252	Das scanCONTROL ist verbunden, `Disconnect();` aufrufen

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

Erlaubt das setzen der für Discovery-Pakete verwendeten Absender IP-Adresse (wenn z.B. mehrere Netzwerkkarten vorhanden sind).

Um über ein Ethernet-Interface angebundene Geräte zu entdecken wird ein Broadcast- Paket verschickt, auf welche sich alle vorhandenen Geräte melden. Standardmäßig geschieht dies über ein limited broadcast, der über alle vorhandenen Netzwerkkarten verschickt wird. Alternativ kann über diese Funktion ein der limited broadcast auf eine Netzwerkkarte beschränkt werden. Über `nNetworkAddress` wird dabei die zu verwendende Absender-IP-Adresse angegeben.

Diese Funktion wird nur vom Ethernet Interface unterstützt, andere Interface Instanzen liefern den Rückgabewert `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. Verbindungs-Funktionen

Mit Hilfe der Verbindungsfunktionen kann eine Verbindung zu einem ausgewählten scanCONTROL aufgenommen oder beendet werden.

```
int Connect()
```

Verbinden der LLT.dll mit einem ausgewählten scanCONTROL. Wurde vorher kein scanCONTROL mit Hilfe von `SetDeviceInterface` ausgewählt, wird bei einer seriellen Verbindung der Com-Port 1 ausgewählt. Beim Ethernet Interface würde eine Verbindung mit der IP-Adresse 0.0.0.0 versucht, welche fehlschlägt. Es ist daher bei Ethernet zwingend erforderlich vorher per SetDeviceInterface eine gültige IP-Adresse anzugeben.

Der Rückgabewert kann einer der allgemeinen oder der folgenden Rückgabewerte sein (siehe Kapitel 2.2 „Allgemeine Rückgabewerte der Funktionen“):

Konstante für den Rückgabewert	Wert	Beschreibung
`ERROR_CONNECT_LLT_COUNT`	-300	Es ist kein scanCONTROL am Computer angeschlossen oder der Treiber ist nicht korrekt installiert
`ERROR_CONNECT_SELECTED_LLT`	-301	Das gewählte Interface ist nicht verfügbar -> ein neues Interface mit `SetDeviceInterface` wählen
`ERROR_CONNECT_ALREADY_CONNECTED`	-302	Mit dieser ID ist schon ein scanCONTROL verbunden
`ERROR_CONNECT_LLT_NUMBER_ALREADY_USED`	-303	Das gewünschte scanCONTROL wird schon von einer anderen Instanz verwendet -> mit `SetDeviceInterface` ein anderes scanCONTROL auswählen
`ERROR_CONNECT_SERIAL_CONNECTION`	-304	Es konnte sich nicht per serieller Schnittstelle mit dem scanCONTROL verbunden werden -> mit `SetDeviceInterface` ein anderes scanCONTROL auswählen
`ERROR_GETDEVINTERFACES_INTERNAL`	-253	Bei der Abfrage der angeschlossenen scanCONTROL ist ein Fehler aufgetreten

```
int Disconnect()
```

Trennen einer Verbindung zu einem scanCONTROL.
Bei einem Disconnect bleiben alle Parameter die im scanCONTROL eingestellt wurden erhalten, außer die „Profile config“ (siehe Kapitel 2.9.6), die „Packet size“ (siehe Kapitel 2.9.5) und der „Buffer count“ im Treiber (siehe Kapitel 2.9.2).

2.7. Identifikations-Funktionen

Funktionen zum identifizieren des scanCONTROL.

2.7.1. GetDeviceName

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

Abfragen des Gerätenamens und des Herstellernamens des scanCONTROL's. Der Gerätenamen ist zum Beispiel „LLT2800-100(000)v17-C2" oder „scanCONTROL 2700-100(000)v20-C2". In dem Gerätenamen sind der Messbereich (in diesem Fall 100mm), die Option (000) und die Softwareversion des scanCONTROL codiert.

Wird einer der Namen nicht benötigt, kann anstatt eines Zeigers auf den Puffer auch eine `NULL` übergeben werden.

Der Rückgabewert kann einer der allgemeinen oder der folgenden Rückgabewerte sein (siehe Kapitel 2.2 „Allgemeine Rückgabewerte der Funktionen"):

Konstante für den Rückgabewert	Wert	Beschreibung
ERROR_GETDEVICENAME_SIZE_TOO_LOW	-1	Die Größe eines der Puffers ist zu klein
ERROR_GETDEVICENAME_NO_BUFFER	-2	Es wurde kein Puffer übergeben

```
char DeviceName[100];

memset(DeviceName, 0, sizeof(DeviceName));
if(pInterfaceLLT->GetDeviceName(DeviceName, sizeof(DeviceName),
                                NULL, NULL) > GENERAL_FUNCTION_NOT_AVAILABLE)
{
  //Verarbeiten des Geraetenamens
}
```

2.7.2. GetLLTVersions

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

Abfragen der Software-Versionen des scanCONTROL. Sie werden automatisch aus dem Gerätenamen extrahiert.

2.7.3. GetLLTType

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

Abfragen von Messbereich und Type des scanCONTROL. Sie werden automatisch aus dem Gerätenamen extrahiert. Wird der Wert `StandardType` zurückgegeben konnte der Typ nicht aus dem Namen extrahiert werden. Bitte wenden Sie sich an die Micro-Epsilon um eine aktuelle DLL zu bekommen. TScannerType ist in „scanControlDataTypes.h“ deklariert.

TScannerType	**Wert**	**scanCONTROL Type**	**Messbereich**
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
scanCONTROL30xx_25	4000	30xx	25 mm
scanCONTROL30xx_50	4001	30xx	50 mm

2.8. Eigenschafts-Funktionen

Mit den Eigenschafts-Funktionen können die verschiedenen Eigenschaften des scanCONTROL gelesen oder geschrieben werden. Es gibt eine `GetFeature`- und eine `SetFeature`-Funktion. Über einen Parameter kann die zu lesende oder schreibende Eigenschaft ausgewählt werden.

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

Der Rückgabewert kann einer der allgemeinen oder der folgenden Rückgabewerte sein (siehe Kapitel 2.2 „Allgemeine Rückgabewerte der Funktionen“):

Konstante für den Rückgabewert	Wert	Beschreibung
ERROR_SETGETFUNCTIONS_WRONG_FEATURE_ADRESS	-155	Die Adresse der gewählten Eigenschaft ist falsch

In folgender Tabelle sind alle verfügbaren Eigenschaften aufgeführt:

Konstante für die Eigenschaft	Adresse	Beschreibung
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
FEATURE_FUNCTION_PEAKFILTER_WIDTH	0xf0b02000	Peakfilter Width
FEATURE_FUNCTION_PEAKFILTER_HEIGHT	0xf0b02004	Peakfilter Height
FEATURE_FUNCTION_FREE_MEASURINGFIELD_Z	0xf0b02008	Free Field z Range
FEATURE_FUNCTION_FREE_MEASURINGFIELD_X	0xf0b0200c	Free Field x Range
FEATURE_FUNCTION_RANGE_SCALE	0xf0a00000	Measuring Range Scale
FEATURE_FUNCTION_RANGE_OFFSET	0xf0a0000a	Measuring Range Offset
FEATURE_FUNCTION_IMAGEFEATURESENABLE	0xf0b02100	
FEATURE_FUNCTION_FREEMEASURINGFIELD2_Z	0xf0b02104	
FEATURE_FUNCTION_FREEMEASURINGFIELD2_X	0xf0b02108	
FEATURE_FUNCTION_REGIONOFNOINTEREST_Z	0xf0b0210c	
FEATURE_FUNCTION_REGIONOFNOINTEREST_X	0xf0b02110	
FEATURE_FUNCTION_AUTOMATICSHUTTERREFERENCEREGION_Z	0xf0b02114	

FEATURE_FUNCTION_AUTOMATICSHUTTERREFERENCEREGION_X	0xf0b02118	
FEATURE_FUNCTION_AUTOMATICSHUTTERLIMITS	0xf0f00834	

Ob die einzelnen Eigenschaften im angeschlossenen scanCONTROL zur Verfügung stehen kann mit Hilfe der Inquiry-Eigenschaften abgefragt werden. Diese Inquiry-Eigenschaften geben die allgemeine Verfügbarkeit, den jeweiligen minimalen und maximalen einstellbaren Wert und die Verfügbarkeit einer automatischen Regelung zurück.

Konstante für die Inquiry-Eigenschaft	**Adresse**	**Beschreibung**
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	Analog frequency
INQUIRY_FUNCTION_ANALOGOUTPUTMODES	0xf0f00520	Analog 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

Der Wert der Inquiry-Eigenschaften kann anhand folgender Tabelle interpretiert werden. Eigenschaften die nicht Verfügbar sind können nicht ausgelesen oder beschrieben werden. Der Wert einer Eigenschaft muss immer innerhalb der, über die Inquiry-Eigenschaften ausgelesenen, Grenzwerte liegen. Höhere Werte werden ignoriert und können zu Fehlfunktionen führen.

Bit	**Beschreibung**
11..0	Maximal einstellbarer Wert
23..12	Minimal einstellbarer Wert
25	Automatische Regelung verfügbar
31	Verfügbarkeit der Eigenschaft

```
DWORD nValue;

//Abfragen der Inquiry-Eigenschaft fuer die Shutter time
if(pInterfaceLLT->GetFeature(INQUIRY_FUNCTION_SHUTTERTIME, &nValue)) ==
GENERAL_FUNCTION_OK)
{
  //Dekodieren der einzelnen Eigenschaften
  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

Lesen der Seriennummer. Diese Eigenschaft kann nur gelesen werden.

2.8.2. Laser power

Abfragen oder Setzen der Laserleistung. Der Laser kann ausgeschaltet, mit reduzierter Leistung eingeschaltet oder mit voller Leistung eingeschaltet werden. Je nach Gerätetyp und Option kann die Polarität der externen Schutzabschaltung eingestellt werden (detaillierte Information zur Eigenschaft im Kapitel „Laser Power“ im „OpManPartB.html“).

Das erste Profil nach dem Umschalten der Laserleistung kann korrupt sein. Es ist daher sinnvoll erst das zweite Profil danach zu verarbeiten.

2.8.3. Measuring field

Abfragen oder Setzen des Messfeldes. Hier wird entweder eines der vordefinierten Messfelder eingestellt oder die erweiterte Messfeldeinstellung aktiviert (detaillierte Information zur Eigenschaft im Kapitel „Measuring Field“ im „OpManPartB.html“). Für weitere Informationen siehe Kapitel 8.2 „Messfeldauswahl und Kalibrierung“ und 11.3 „Unterstützte Messfelder“ in der Bedienungsanleitung .
Die Auswahl des Messfeldes hat Einfluss auf die maximale Profilfrequenz des Sensors (siehe Kapitel „Maximum Frequencies of Profile Measurements“ in der „QuickReference.html“).

Das erste Profil nach dem Umschalten des Messfeldes kann korrupt sein. Es ist daher sinnvoll erst das zweite Profil danach zu verarbeiten.

2.8.4. Trigger

Abfragen oder Setzen der Triggerkonfiguration. Sensoren können intern oder über einen externen Eingang getriggert werden (detaillierte Information zur Eigenschaft im Kapitel „External Trigger input“ im „OpManPartB.html“).

Das erste Profil nach dem Umschalten des Trigger-Modes kann korrupt sein. Es ist daher sinnvoll erst das zweite Profil danach zu verarbeiten.

2.8.5. Shutter time

Abfragen oder Setzen der Belichtungszeit. Sie kann zwischen 1 und 4095 liegen. Ein Zählwert entspricht dabei 10 µs. Aus der Summe von Belichtungszeit und Totzeit (siehe Kapitel 2.8.6 „Idle time“) kann die Profilfrequenz berechnet werden (detaillierte Information zur Eigenschaft im Kapitel „Shutter Time“ im „OpManPartB.html“)

Der Bildsensor unterstützt überlappendes auslesen. Wenn die Belichtungszeit zum Auslesen des Sensors reicht, kann die Totzeit auf 30 µs verkleinert werden.

Die erreichbare maximale Profilrate hängt von der Auflösung (Anzahl der Punkte pro Profil) und dem Messfeld ab. Mehr Details sind im Kapitel 8.2 „Messfeldauswahl und Kalibrierung“ und Kapitel 11.3 „Unterstützte Messfelder“ in der Bedienungsanleitung zu finden. Eine Tabelle mit den zu erreichenden Profilraten bei den unterschiedlichen Auflösungen und Paketgrößen ist in der „QuickReference.html“ zu finden.

Profilrate = 1000 / (Belichtungszeit in ms + Totzeit in ms)

Die Sensoren unterstützen auch eine automatische Belichtungsregelung. Sie regelt die Belichtungszeit in Abhängigkeit vom Messobjekt. Die eingestellte Belichtungszeit gibt dabei

einen Vorgabewert vor, welcher verwendet wird wenn kein Messobjekt gesehen wird. Bei der automatischen Belichtungszeitregelung ändert sich nicht die Profilrate.

2.8.6. Idle time

Abfragen oder Setzen der Totzeit. Sie kann zwischen 1 und 4095 liegen. Ein Zählwert entspricht dabei 10 µs. Aus der Summe von Totzeit und Belichtungszeit (siehe Kapitel 0 „Shutter time“) kann die Profilfrequenz berechnet werden.

Profilrate = 1000 / (Belichtungszeit in ms + Totzeit in ms)

Ist die automatische Belichtungszeitregelung aktiv (siehe Kapitel 0 „Shutter time“) wird die Totzeit automatisch angepasst, damit die Profilfrequenz stabil bleibt.

2.8.7. Processing profile data

Abfragen oder Setzen der Einstellungen für die Profilverarbeitung (detaillierte Information zur Eigenschaft im Kapitel „Processing of Profile Data“ im „OpManPartB.html“).

2.8.8. Threshold

Schwelle zur Auswahl von Reflektionen. Bei Targets mit mehreren Reflektionen kann das erhöhen der Schwelle zu besseren Ergebnissen führen (detaillierte Information zur Eigenschaft im Kapitel „Threshold“ im „OpManPartB.html“).

2.8.9. Maintenance functions

Abfragen oder Setzen interner Einstellungen. Die meisten dieser Einstellungen sollten unverändert verwendet werden. (detaillierte Information zur Eigenschaft im Kapitel „Maintenance functions“ im „OpManPartB.html“).
Nur die Bits für den Encoder-Eingang können allgemein benutzt werden. Mit Ihnen kann der interne Encoder-Zähler aktiviert werden. Beim Aktivieren wird er gleichzeitig gelöscht. Der Zählerstand wird in jedem Profil übermittelt.

```
#include <vector>
//Aktivieren des internen Zaehlers, Aufloesung setzen
pInterfaceLLT->SetFeature(FEATURE_FUNCTION_MAINTENANCEFUNCTIONS,
                          0x00000010);
pInterfaceLLT->SetResolution(256);

//Aktivieren der Profiluebertragung und einen Moment warten (auf Profile)
pInterfaceLLT->TransferProfiles(NORMAL_TRANSFER, true);
Sleep(500);

//Erstellen eines Puffers fuer ein Profil in PURE_PROFILE Mode und abholen
//eines Profils
unsigned int ProfSize = 256*4+16;
std::vector<unsigned char> vProfile(ProfSize);
if(pInterfaceLLT->GetActualProfile(&vProfile[0], ProfSize, PURE_PROFILE,
                                   NULL) != ProfSize)
{
  //Das scanCONTROL sendet keine Profile
  return;
}
unsigned int LastCount = 0, OverflowCount = 0, TempCount;

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

//Testen ob der 16 bit Zähler uebergelaufen ist
if(TempCount < (LastCount & 0x0000ffff))
{
  OverflowCount += 1;
}
//Erstellen des echten Zählerstandes mit allen Ueberlaeufen
LastCount = TempCount + 0x00010000 * OverflowCount;
```

2.8.10. Analog frequency

Abfragen oder Setzen der Frequenz für den Analogausgang. Die Frequenz kann zwischen 0 und 150 eingestellt werden, wobei der Zählwert der Frequenz in kHz entspricht. Bei einer Einstellung von 0 kHz wird der Analogausgang abgeschaltet, was bei Profilfrequenzen größer 500 Hz empfehlenswert ist, um einen Überlauf bei der Analogausgabe zu vermeiden.

Dieser Parameter steht nur für scanCONTROL 28xx zur Verfügung. Detaillierte Information zur Eigenschaft stehen im Kapitel „Speed of Analogue Outputs“ im „OpManPartB.html“.

2.8.11. Analog output modes

Einstellen der Analog output modes. Es können z.B. die Spannungsbereiche und die Polarität der analogen Ausgänge umgeschaltet werden.

Dieser Parameter steht nur für scanCONTROL 28xx zur Verfügung. Detaillierte Information zur Eigenschaft stehen im Kapitel „Analogue Output Mode“ im „OpManPartB.html“.

2.8.12. CMMTrigger

Konfiguration des optionalen CMM-Triggers. Die Konfiguration des CMM-Triggers erfolgt durch mehrere Schreibzugriffe auf dieses Register. Zurückgelesen werden kann nur der letzte Wert. Detaillierte Information zur Eigenschaft stehen im Kapitel „cmmTrigger“ im „OpManPartB.html“.

2.8.13. Rearrangement profile

Im Container-Mode werden mehrere Profile zu einem Container/Bild zusammengefasst. Zusätzlich können die Daten im Container so angeordnet werden, dass sie direkt als Bild interpretiert werden können. Detaillierte Information zum Containermode stehen im Kapitel „Container Mode for Transmission“ im „OpManPartB.html“.

Beispiele zum Übertragen von Containern sind im Kapitel 4 „LLT2800Samples“ zu finden.

Der Rückgabewert kann einer der allgemeinen oder der folgenden Rückgabewerte sein (siehe Kapitel 2.2 „Allgemeine Rückgabewerte der Funktionen“):

Konstante für den Rückgabewert	Wert	Beschreibung
`ERROR_SETGETFUNCTIONS_REARRANGEMENT_PROFILE`	-159	Der `Rearrangement`-Parameter ist falsch

Die Breite eines Bildes zu errechnet sich folgendermaßen:

Breite = Anzahl der Eigenschaften (Bit 0 bis 8) * 2 * Resolution * Anzahl der Streifen

Die Anzahlen der Eigenschaften und Streifen bezieht sich nur auf die auszugebenden und nicht auf die vorhandenen.
Nach dem Schreiben dieser Eigenschaft berechnet die LLT.dll selbständig die nötige Breite des Bildes und stellt sie ein. Die Breite kann wie in Kapitel 2.9.8 „Profile container size“ beschrieben ausgelesen werden.
Die Paketgröße zum Übertragen der umgewandelten Profile muss immer einem ganzzahligen vielfachen der Spaltenbreite entsprechen. Die LLT.dll stellt beim Starten der Übertragung automatisch eine passende Paketgröße, die maximal der aktuell eingestellten Paketgröße entspricht, ein.

Sollen nur aufeinander folgende Profile ohne Umsortieren zu einem Container verbunden werden, ist nur das Bitfeld ‚Punkte pro Profil’ anzugeben. Mit der Funktion `SetProfileContainerSize(0, nProfileCount)` ist dann die Anzahl der als Container zu übertragenden Profile anzugeben.

2.8.14. Profile filter

Mit Hilfe des Profil Filters können einfache Filter direkt im scanCONTROL auf ein Profil angewandt werden. Dadurch können z.B. ausreißende Punkte aussortiert werden (detaillierte Information zur Eigenschaft im Kapitel „Profile Filter“ im „OpManPartB.html“).

2.8.15. Interface function/Port configuration

Bestimmt die Funktionen der digitalen Schnittstelle. Das Interface kann für die serielle Kommunikation oder beispielsweise den externen Trigger verwendet werden. Die Funktion des Interfaces muss mit anderen Einstellungen überein stimmen. Zum Beispiel muss für den externen Trigger das Interface auf `RS422_INTERFACE_FUNCTION_TRIGGER` gestellt und der externe Trigger aktiviert werden.

Das Interface hat zusätzlich einen Automatic Mode (als Standard). Das Interface steht nach dem booten auf serieller Kommunikation (115200 Baud) und wechselt zu `RS422_INTERFACE_FUNCTION_TRIGGER` Mode wenn der externe Trigger aktiviert wird. Das

Interface bleibt bei `RS422_INTERFACE_FUNCTION_TRIGGER` solange bis es manuell geändert wird oder bis der Sensor neu gebootet wird.

Detaillierte Information zur Schnittstellenkonfiguration stehen im Kapitel „Serial Interface Function“ bzw. „Multi-Function Port Configuration“ im „OpManPartB.html“.

2.8.16. Packet delay

Gibt die Verzögerung zwischen zwei Ethernet-Paketen in Mikrosekunden an. Der Wert darf zwischen 0 und 1000µs liegen.
Diese Funktion steht nur bei scanCONTROL mit Ethernet-Schnittstelle zur Verfügung. Durch das Packet delay kann ein Switch, an den mehrere scanCONTROL angeschlossen sind, entlastet werden.

2.8.17. Peakfilter Breite und Höhe

Ermöglicht das Filtern von Peaks nach vorgegeben Eigenschaften (detaillierte Information zur Eigenschaft im Kapitel „Extra Parameter“ im „OpManPartB.html“). Zur Aktivierung der neuen Parameter muss `SetFeature(FEATURE_FUNCTION_SHARPNESS,0)` aufgerufen werden.

```
// 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. Frei definierbares Messfeld X/Z

Dient dem setzen der Parameter für das freie Messfeld (detaillierte Information zur Eigenschaft im Kapitel „Extra Parameter“ im „OpManPartB.html“). Zur Aktivierung der neuen Parameter muss `SetFeature(FEATURE_FUNCTION_SHARPNESS,0)` aufgerufen werden.

```
// 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
```

2.8.19. Aktivieren weiterer Imageparameter

2.8.20. Zweites frei definierbares Messfeld X/Z

2.8.21. Region of No Interest

2.8.22. Referenzregion für die Belichtungsautomatik

2.9. Spezielle Eigenschafts-Funktionen

Mit Hilfe der speziellen Eigenschafts-Funktionen können weitere Eigenschaften des scanCONTROL’s geschrieben und gelesen werden, welche nicht in das Konzept der normalen Eigenschafts-Funktionen passen.

2.9.1. SetFeatureGroup

```
SetFeatureGroup(const DWORD* FeatureAddresses, const DWORD* FeatureValues,
DWORD FeatureCount)
```

Schreibt die Werte die in FeatureValues angegeben werden auf die in FeatureAddresses angegebenen Adressen. Wenn die Adressen zusammenhängend sind werden die Werte als ein Block geschrieben.

2.9.2. Buffer count

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

Abfragen oder Setzen der Pufferanzahl im scanCONTROL-Treiber. Je größer die Anzahl der Puffer ist, desto mehr Profile/Container können zwischengespeichert werden, bevor sie von der LLT.dll abgeholt werden müssen.

Eine hohe Pufferanzahl ist vor allem bei sehr hohen Profilfrequenzen, langsamen Rechnern und/oder Rechnern bei denen mehrere Programme im Hintergrund laufen sinnvoll. Ist die

Pufferanzahl zu gering kommt es zu Ausfällen in der Profilübertragung, sodass die Profilfrequenz sinkt. Standardmäßig sind 20 Puffer eingestellt.
Soll eine Übertragung im Container-Mode gestartet werden ist darauf zu achten das die Anzahl der verwendeten Puffer maximal 4 beträgt. Sonst kann es zu einem sehr langsamen starten der Profilübertragung und eines sehr hohen Speicherverbrauches kommen. Werden sehr große Container verwendet, reichen auch 3 Puffer.

Der Rückgabewert kann einer der allgemeinen oder der folgenden Rückgabewerte sein (siehe Kapitel 2.2 „Allgemeine Rückgabewerte der Funktionen“):

Konstante für den Rückgabewert	Wert	Beschreibung
ERROR_SETGETFUNCTIONS_WRONG_BUFFER_COUNT	-150	Die Anzahl der gewünschten Puffer liegt nicht im Bereich >=2 und <= 200

2.9.3. Main reflection

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

Abfragen oder Setzen des auszugebenden Streifens bei einer Profilkonfiguration von PURE_PROFILE oder QUARTER_PROFILE. Dieser Index wird in der LLT.dll zur Umwandlung von PROFILE in PURE_PROFILE und QUARTER_PROFILE benötigt.
Der Index des auszugebenden Streifens geht von 0 für den 1. Streifen bis 3 für den 4. Streifen.
Eine Beschreibung der Profil-Daten und der Streifen befindet sich in Kapitel 3.1 „Beschreibung der Profil-Daten“.

Diese Funktion steht nur bei einer Ethernetübertragung zur Verfügung.

Der Rückgabewert kann einer der allgemeinen oder der folgenden Rückgabewerte sein (siehe Kapitel 2.2 „Allgemeine Rückgabewerte der Funktionen“):

Konstante für den Rückgabewert	Wert	Beschreibung
ERROR_SETGETFUNCTIONS_REFLECTION_NUMBER_TOO_HIGH	-154	Der Index des auszugebenden Streifens ist größer 3

2.9.4. Max filesize

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

Abfragen oder Setzen der maximalen Dateigröße beim Speichern von Profilen in Byte. Ist diese Größe erreicht stoppt das Speichern.

2.9.5. Packet size

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

Abfragen der minimalen und maximalen Paketgröße der Ethernet Streaming Pakete.
Diese Funktion steht nur bei der Ethernet Übertragung zur Verfügung.

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

Abfragen oder Setzen der aktuellen Paketgröße der Ethernet Streaming Pakete. Diese Paketgröße muss zwischen der minimalen und maximalen Paketgröße liegen. Vom scanCONTROL werden die Paketgrößen 128, 256, 512, 1024, 2048 und 4096 Bytes unterstützt. Pakete größer als 1024 Bytes erfordern bei Ethernet die Unterstützung von Jumbo Frames durch die gesamte Übertragungsstrecke, insbesondere der empfangenden Netzwerkkarte.

Der Rückgabewert kann einer der allgemeinen oder der folgenden Rückgabewerte sein (siehe Kapitel 2.2 „Allgemeine Rückgabewerte der Funktionen“):

Konstante für den Rückgabewert	Wert	Beschreibung
ERROR_SETGETFUNCTIONS_PACKET_SIZE	-151	Die gewünschte Paketgröße wird nicht unterstützt

```
unsigned long MaxPacketSize = 0;

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

2.9.6. Profile config

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

Abfragen oder Setzen der Profilkonfiguration.

ProfileConfig	Wert	Beschreibung
PROFILE	1	Profildaten aller vier Streifen
PURE_PROFILE	2	Reduzierte Profildaten eines Streifens (nur Positions- und Abstands-Werte)
QUARTER_PROFILE	3	Profildaten eines Streifens
PARTIAL_PROFILE	5	Partielles Profile welches per SetPartialProfile eingeschränkt wurde
CONTAINER	1	Container-Daten
VIDEO_IMAGE	1	Video-Bild des scanCONTROL's

Bei einer Verbindung über die serielle Schnittstelle wird nur die Profilkonfiguration PURE_PROFILE unterstützt.
Die Profilkonfiguration gilt gleichzeitig für das Speichern von Profilen und den Callback. Weiterführend siehe Kapitel 3 „Profil-/Container/Video-Übertragung".

Der Rückgabewert kann einer der allgemeinen oder der folgenden Rückgabewerte sein (siehe Kapitel 2.2 „Allgemeine Rückgabewerte der Funktionen"):

Konstante für den Rückgabewert	Wert	Beschreibung
ERROR_SETGETFUNCTIONS_WRONG_PROFILE_CONFIG	-152	Die gewünschte Profilkonfiguration steht nicht zur Verfügung

```
TProfileConfig ProfileConfig;

pInterfaceLLT->GetProfileConfig(&ProfileConfig);

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

2.9.7. Resolution

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

Abfragen oder Setzen der Auflösung von Profilen. Die Standard-Auflösungen sind für die verschiedenen scanCONTROL-Varianten in folgender Tabelle dargestellt:

Der Rückgabewert kann einer der allgemeinen oder der folgenden Rückgabewerte sein (siehe Kapitel 2.2 „Allgemeine Rückgabewerte der Funktionen“):

Konstante für den Rückgabewert	Wert	Beschreibung
ERROR_SETGETFUNCTIONS_NOT_SUPPORTED_RESOLUTION	-153	Die gewünschte Auflösung wird nicht unterstützt

Alle möglichen Auflösungen können mit Hilfe der Funktion `GetResolutions` ausgelesen werden.

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

Dieser Funktion muss ein Feld von DWORD-Variablen übergeben werden. In dieses Feld werden dann alle möglichen Auflösungen eingetragen. Zurzeit sind nur maximal 6 verschiedene Auflösungen möglich. Ist die Größe des Feldes zu klein wird `ERROR_SETGETFUNCTIONS_SIZE_TOO_LOW` zurückgegeben, sonst die Anzahl der eingetragenen Auflösungen.

Der Rückgabewert kann einer der allgemeinen oder der folgenden Rückgabewerte sein (siehe Kapitel 2.2 „Allgemeine Rückgabewerte der Funktionen“):

Konstante für den Rückgabewert	Wert	Beschreibung
ERROR_SETGETFUNCTIONS_SIZE_TOO_LOW	-156	Die Größe des übergebenen Feldes ist zu klein

Für die scanCONTROL 28xx gilt: Je größer die Auflösung der Profile ist, desto geringer ist die maximale Profilfrequenz (siehe Kapitel „Maximum Frequencies of Profile Measurements“ in der „QuickReference.html“ für den scanCONTROL2800).

Die Auflösung kann nur dann geändert werden, wenn keine Profile übertragen werden. Außerdem werden bei `SetResolution` alle Einstellungen für das `PartialProfile` gelöscht (siehe Kapitel 2.13 „PartialProfile-Funktionen“).

2.9.8. Profile container size

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

Abfragen der maximalen `ProfileContainerSize` für den Container-Mode. (siehe Kapitel 2.8.13 „Rearrangement profile“).
Ist die maximale Breite 64, so wird der Container-Mode nicht von dem scanCONTROL unterstützt.

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

Abfragen oder Setzen der Größe des Containers für den Container-Mode.
Die Breite wird automatisch beim Aufruf von `SetFeature(FEATURE_FUNCTION_REARRANGEMENT_PROFILE)` gesetzt (siehe Kapitel 2.8.13 „Rearrangement profile“).
Die Höhe kann frei zwischen 0 und der maximal möglichen Höhe gewählt werden und entspricht der Anzahl von Profilen in dem Container übertragen werden.
Die Container-Höhe darf nicht höher als die dreifache Profilrate (siehe Kapitel 0 „
Shutter time“) sein, da sichergestellt sein muss, dass mindestens alle 3 Sekunden ein Container übertragen wird. Sonst kann es zu erheblichen ausfällen kommen.

Werden nicht alle Parameter benötigt, kann für die nicht benötigten alternativ auch `NULL` übergeben werden.

Der Rückgabewert kann einer der allgemeinen oder der folgenden Rückgabewerte sein (siehe Kapitel 2.2 „Allgemeine Rückgabewerte der Funktionen“):

Konstante für den Rückgabewert	Wert	Beschreibung
`ERROR_SETGETFUNCTIONS_WRONG_PROFILE_SIZE`	-157	Die Größe für den Container ist falsch
`ERROR_SETGETFUNCTIONS_MOD_4`	-158	Die Container-Breite ist nicht durch 4 Teilbar

Ist „Verbinden von aufeinanderfolgenden Profilen“ aktiviert (siehe Kapitel 2.8.13 „Rearrangement profile“) muss die Höhe * Breite eines Bildes ein ganzzahliges vielfaches von 16384 sein. Wird versucht einen anderen Höhenwert einzustellen, wird die Höhe automatisch auf den nächsten passenden Wert gesetzt. Zusätzlich wird der Fehlerwert `GENERAL_FUNCTION_CONTAINER_MODE_HEIGHT_CHANGED` ausgegeben um auf die Änderung aufmerksam zu machen.

2.9.9. Laden und Speichern der Usermodes

Laden und Speichern der Usermodes. In einem Usermode können alle Einstellungen eines scanCONTROL gespeichert werden, so dass nach einem Reset oder Neustart sofort alle Einstellungen wieder aktiv sind. Dies ist vor allem bei Postprocessing-Anwendungen sinnvoll. Das Laden der Usermodes kann nicht während einer aktiven Profil/Container-Übertragung durchgeführt werden.
Usermode 0 kann nur geladen werden, da er die Standardeinstellungen enthält.

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

Auslesen der verfügbaren Usermodes. `pActualUserMode` ist ein Pointer zu einem Feld mit Integer werten für die verfügbaren Usermodes, `pUserModeCount` ist die Größe des übergebenen Feldes. Die scanCONTROL 28xx unterstützen 4 und die scanCONTROL 26xx/27xx/29xx 16 Usermodes.

```
unsigned int UserModes[10];

//Abfragen der vorhandenen Usermodes
if(pInterfaceLLT->GetActualUserMode(&UserModes, sizeof(ErrorString)) >
GENERAL_FUNCTION_NOT_AVAILABLE)
{
  //Auswertung des Usermodes
}
```

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

Laden oder Speichern eines Usermodes. Ist `nWrite` 0 wird der mit `nUserMode` angegebene Usermode geladen, ansonsten werden die aktuellen Einstellungen unter diesem Usermode gespeichert. Nach dem Laden eines Usermodes muss ein Disconnect und wieder Connect ausgeführt werden.

Konstante für den Rückgabewert	Wert	Beschreibung
ERROR_SETGETFUNCTIONS_USER_MODE_TOO_HIGH	-160	Die angegebene Usermode-Nummer steht nicht zur Verfügung
ERROR_SETGETFUNCTIONS_USER_MODE_FACTORY_DEFAULT	-161	Usermode 0 kann nicht überschrieben werden (Standardeinstellungen)

```
//Speichern des Usermodes 1
if(pInterfaceLLT->ReadWriteUserModes(1, 1) >
GENERAL_FUNCTION_NOT_AVAILABLE)
{
  //Usermode wurde erfollgreich gespeichert
}

//Laden des Usermodes 1
if(pInterfaceLLT->ReadWriteUserModes(0, 1) >
GENERAL_FUNCTION_NOT_AVAILABLE)
{
  //Usermode wurde erfollgreich geladen
}
```

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

Nur sinnvoll bei Ethernet Sensoren, da hiermit die IP-Einstellungen gespeichert werden.

2.9.10. Ethernet Heartbeat timeout

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

Setzen und Auslesen des Heartbeat Timeouts in Millisekunden zur Überwachung der Kommunikations-Schnittstelle zwischen LLT.dll und dem scanCONTROL. Der eigentliche Timeout-Wert liegt dreimal höher als der eingestellte Heartbeat Timeout.
Läuft der Timeout ohne den Heartbeat ab, wird Kommunikation automatisch vom Sensor aus abgebrochen. Die LLT.dll schickt in diesem Fall eine Message an ein registriertes Fenster (siehe Kapitel 2.10.2 Message). Das Ändern von Parametern gibt danach Fehlerwerte zurück.
Der Heartbeat Timeout kann zwischen 500 und 1.000.000.000 ms liegen.

Diese Funktion steht nur bei scanCONTROL mit Ethernet-Schnittstelle zur Verfügung.

Das Heartbeat Timeout muss nur dann geändert werden, wenn die Applikation, welche die LLT.dll verwendet, debuggt werden soll. Da in diesem Fall die LLT.dll angehalten wird und keinen weiteren Heartbeat schicken kann. Hierfür empfehlen wir einen Wert von 200000ms.

Der Rückgabewert kann einer der allgemeinen oder der folgenden Rückgabewerte sein (siehe Kapitel 2.2 „Allgemeine Rückgabewerte der Funktionen“):

Konstante für den Rückgabewert	Wert	Beschreibung
ERROR_SETGETFUNCTIONS_HEARTBEAT_TOO_HIGH	-162	Der Parameter für den Heartbeat Timeout ist zu groß

2.9.11. HoldBuffersForPolling

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

Abfragen oder Setzen der Anzahl von Profilen, welche in der LLT.dll für das Abholen mit GetActualProfile (siehe Kapitel 2.11.5 „Get actual profile“) vorgehalten werden sollen.
Je größer die Anzahl ist, desto mehr Profile werden zwischengespeichert und die Häufigkeit von Profilausfällen bei dem Abholen mit GetActualProfile wird verringert.

Bit	Funktion
7..0	Anzahl der vorgehalten Profile

Die Anzahl kann maximal halb so groß wie die Anzahl der Puffer im Treiber sein (siehe Kapitel 2.9.2 „Buffer count“).

Der Rückgabewert kann einer der allgemeinen oder der folgenden Rückgabewerte sein (siehe Kapitel 2.2 „Allgemeine Rückgabewerte der Funktionen“):

Konstante für den Rückgabewert	Wert	Beschreibung
ERROR_SETGETFUNCTIONS_WRONG_BUFFER_COUNT	-150	Die Anzahl der gewünschten Puffer liegt nicht im Bereich >=2 und <= 200

2.10. Register-Funktionen

Funktionen zum Registrieren eines Callbacks für Profile und einer Message für Fehlermeldungen.

2.10.1. Callback

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

Registrieren eines Callbacks vom Typ `TNewProfile_s` oder `TNewProfile_c` welcher immer dann aufgerufen wird wenn ein neues Profil/Container empfangen wurde. Dieser Callback muss nur dann registriert werden, wenn er verwendet werden soll.
Mit Hilfe des Parameters `CallbackType` wird die Aufrufkonvention des Callbacks festgelegt. Stimmt die Aufrufkonvention nicht mit der Aufrufkonvention der eigenen Callback-Funktion überein kann es zu Programmabstürzen kommen.

CallbackType	Wert	Beschreibung
`STD_CALL`	0	Der Callback arbeitet mit `stdcall` (`TNewProfile_s`)
`C_DECL`	1	Der Callback arbeitet mit `cdecl` (`TNewProfile_c`)

Der Parameter `pUserData` dient zur Unterscheidung von welcher scanCONTROL-Instanz der Callback aufgerufen wurde. Es kann z.B. der Pointer zu einer Klasse oder eine Nummer übergeben werden.
Wird dieser Funktion `NULL` als `pLLTProfileCallback` übergeben, wird der Callback wieder deaktiviert. Wobei darauf zu achten ist, das die Aufrufkonvention auch hier korrekt mit angegeben werden muss.

Weitere Informationen zu diesem Thema sind in den Kapiteln 2.11 „Profilübertragungs-Funktionen“ und 3 „Profil-/Container/Video-Übertragung“ zu finden.

```
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);
```

Diese Callbacks werden, wenn sie registriert wurden, nach dem Empfangen eines Profils/Containers aufgerufen und besitzt als Parameter einen Pointer auf die Profil-/Container-Daten, die dazugehörige Größe des Datenfeldes und einen `pUserData`-Parameter. Mit dem `pUserData`-Parameter kann zum Beispiel unterschieden werden von welchem scanCONTROL das Profil / der Container stammt.

Der Callback ist für die Verarbeitung von Profilen/Containern mit einer hohen Profilfrequenz gedacht. Innerhalb des Callback können die Profile/Container in einen Puffer für eine spätere oder zum Callback synchrone oder asynchrone Verarbeitung kopiert werden. Eine Verarbeitung innerhalb des Callbacks ist nicht zu empfehlen, da für die Zeit die der Callback zur Verarbeitung benötigt die LLT.dll keine neuen Profile/Container vom Treiber abholen kann. Unter Umständen kann es dadurch zu Profil-/Container-Ausfällen kommen.
Die Profil-/Container-Daten in dem vom Callback übergebenen Puffer dürfen nicht verändert werden.

Die Profilkonfiguration des Pointers kann über die Funktion `SetProfileConfig` eingestellt werden und gilt gleichzeitig auch für das Speichern von Profilen/Containern (siehe Kapitel 2.9.6 „Profile config“).

```
#include <vector>
HANDLE hProfileEvent;
//Anlegen eines Puffers fuer ein Profil mit der maximalen Profilgroesse
std::vector<unsigned char> ProfileBuffer(1024*64);

void GetProfileFromCallback()
{
  //Erstellen eines Events
  hProfileEvent = CreateEvent(NULL, true, false, "ProfileEvent");

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

  //Aktivieren der Profiluebertragung
  if(pInterfaceLLT->TransferProfiles(NORMAL_TRANSFER, true)
                                        <= GENERAL_FUNCTION_NOT_AVAILABLE)
    return;

  //Warten auf ein Event vom Profil-Callback
  if(WaitForSingleObject(hProfileEvent, 1000) != WAIT_OBJECT_0)
  {
    //Fehler beim warten auf den Callback
    return;
  }
  //Auswerten des Profiles

  //Deaktivieren der Profiluebertragung
  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)
  {
    //Wenn die Profil kleiner gleich der Puffergroesse ist:
    //Kopieren des Profiles in den Puffer
    memcpy(&ProfileBuffer[0], pData, nSize);
    SetEvent(hProfileEvent);
  }
}
```

2.10.2. Message

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

Registrieren einer Fehler-Message welche bei Fehlern gesendet wird.

Fehlercode im LPARAM	Wert	Beschreibung
ERROR_SERIAL_COMM	1	Fehler während der seriellen Datenübertragung. Eventuell ist die Profilfrequenz zu hoch.
ERROR_SERIAL_LLT	7	scanCONTROL konnte Kommando nicht verstehen oder es wurde ein Parameter außerhalb des Gültigkeitsbereiches gesendet.
ERROR_CONNECTIONLOST	10	Die Verbindung zum scanCONTROL wurde unterbrochen (scanCONTROL wurde Abgeschaltet, reseted oder das Ethernet-Kabel wurde entfernt). Bitte senden sie ein „Disconnect“ aus um sich neu verbinden zu können. Diese Message wird nur bei einer Verbindung über Ethernet gesendet.
ERROR_STOPSAVING	100	Das Speichern von Profilen ist beendet (maximale Dateigröße erreicht).

2.11. Profilübertragungs-Funktionen

Beschreibung der Funktionen für die Profilübertragung. Genauer wird darauf im Kapitel 3 „Profil-/Container/Video-Übertragung“ eingegangen.

Beispiele zum Übertragen von Profilen mit unterschiedlichen Profilkonfigurationen und das Umrechnen der Profildaten in Millimeter sind im Kapitel 4 „LLT2800Samples“ zu finden.

2.11.1. Transfer profiles

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

Funktion zum Starten oder Beenden der Profilübertragung.
Der Parameter `TransferProfileType` gibt an, ob eine kontinuierliche oder eine bedarfsmäßige Übertragung aktiviert werden soll.

TransferProfileType	Wert	Beschreibung
NORMAL_TRANSFER	0	Aktivieren einer kontinuierlichen Übertragung von Profilen
SHOT_TRANSFER	1	Aktivieren einer Bedarfsmäßigen Übertragung von Profilen (die Übertragung wird immer per `MultiShot` aktiviert)
NORMAL_CONTAINER_MODE	2	Aktivieren einer kontinuierlichen Übertragung im Container-Mode
SHOT_CONTAINER_MODE	3	Aktivieren einer Bedarfsmäßigen Übertragung im Container-Mode (die Übertragung wird immer per `MultiShot` aktiviert)

Der Parameter `nEnable` gibt an, ob eine Übertragung gestartet oder beendet werden soll. Nach dem starten einer Übertragung kann es bis zu 100 ms dauern ehe die ersten Profile/Container per Callback ankommen oder per `GetActualProfile` abgeholt werden können.
Wird eine Übertragung beendet, wartet die Funktion automatisch, bis der Treiber alle Puffer zurückgegeben hat.

Soll eine Übertragung im Container-Mode gestartet werden (siehe Kapitel 2.8.13 „Rearrangement profile“) ist darauf zu achten das die Anzahl der verwendeten Puffer maximal 4 beträgt (siehe Kapitel 2.9.2 „Buffer count“). Sonst kann es zu einem sehr langsamen starten der Profilübertragung und eines sehr hohen Speicherverbrauches kommen. Werden sehr große Container verwendet, reichen auch 3 Puffer.

Der Rückgabewert ist die Größe eines Profiles/Containers. Ist die Größe kleiner oder gleich `GENERAL_FUNCTION_NOT_AVAILABLE` ist es einer der allgemeinen oder der folgenden Rückgabewerte (siehe Kapitel 2.2 „Allgemeine Rückgabewerte der Funktionen“):

Konstante für den Rückgabewert	Wert	Beschreibung
`ERROR_PROFTRANS_PACKET_SIZE_TOO_HIGH`	-107	Die Paketgröße ist größer als die verfügbare -> mit `SetPacketSize` eine niedrigere Paketgröße einstellen
`ERROR_PROFTRANS_CREATE_BUFFERS`	-108	Die Puffer für den Treiber konnten nicht ordnungsgemäß angelegt werden -> ev. PC neu starten
`ERROR_PROFTRANS_WRONG_PACKET_SIZE_FOR_CONTAINER`	-109	Es kann für die gewählten Container-Einstellungen keine passende Paketgröße gefunden werden. Bitte erhöhen Sie die Paketgröße (siehe Kapitel 2.9.5 Packet size auf Seite 24)

2.11.2. Transfer video stream

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

Funktion zum Starten oder Beenden der Übertragung von Video-Bildern des Bildsensors.
Der Parameter `TransferVideoType` gibt den verwendeten Übertragungsmode an.

TransferVideoType	Wert	Beschreibung
`VIDEO_MODE_0`	0	Verkleinertes Bild der Matrix
`VIDEO_MODE_1`	1	Vollständiges Bild der Matrix

Die Größe der übertragenen Bilder von der Matrix werden in die Parametern `pWidth` und `pHeight` kopiert.

Zu beachten ist, dass die Video-Bilder nur mit maximal 25 Bildern pro Sekunde übertragen werden können. Die Werte für die `Shutter time` und `Idle time` müssen dem entsprechend geändert werden (siehe Kapitel 0 „
Shutter time“ und Kapitel 2.8.6 „Idle time“).
Die `Packet size` muss für die scanCONTROL 28xx 4096 betragen und für die scanCONTROL 26xx/27xx/29xx größer oder gleich 2048 sein.

Video-Bilder können nur mit `GetActualProfile` abgeholt. Per `Callback` stehen sie nicht zur Verfügung.
Video Bilder können nur einzeln als Bitmap gespeichert werden (siehe Kapitel 2.16.1. „Speichern von Profilen“).

Der Rückgabewert ist die Größe eines Video Bildes. Ist die Größe kleiner oder gleich `GENERAL_FUNCTION_NOT_AVAILABLE` ist es einer der allgemeinen oder der folgenden Rückgabewerte (siehe Kapitel 2.2 „Allgemeine Rückgabewerte der Funktionen“):

Konstante für den Rückgabewert	Wert	Beschreibung
`ERROR_PROFTRANS_PACKET_SIZE_TOO_HIGH`	-107	Die Paketgröße ist größer als die verfügbare -> mit `SetPacketSize` eine niedrigere Paketgröße einstellen
`ERROR_PROFTRANS_CREATE_BUFFERS`	-108	Die Puffer für den Treiber konnten nicht ordnungsgemäß angelegt werden -> ev. PC neu starten

2.11.3. Multi shot

Mit Hilfe der Funktionen `MultiShot` ist es möglich nur ein Profil/Container oder eine bestimmte Menge an Profilen/Containern zu übertragen.
Bei dem scanCONTROL 28xx steht dieses Feature steht erst ab einer DSP-Firmwareversion von 10 zur Verfügung.

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

Anfordern von mehreren Profilen/Containern. Die Anzahl der Profile/Container wird in dem Parameter `nCount` übergeben. Es können zwischen 1 und 65535 Profile/Container angefordert werden.
Wird 0 übergeben wird der interne Zähler für die noch ausstehenden Profile/Container wieder auf 0 gesetzt -> es kann eine neue Anforderung ausgelöst werden.

Um MultiShot nutzen zu können muss beim starten der Übertragung der `SHOT_TRANSFER`- oder der `SHOT_CONTAINER_MODE`-Mode ausgewählt worden sein.

Der Rückgabewert kann einer der allgemeinen oder der folgenden Rückgabewerte sein (siehe Kapitel 2.2 „Allgemeine Rückgabewerte der Funktionen“):

Konstante für den Rückgabewert	Wert	Beschreibung
`ERROR_PROFTRANS_SHOTS_NOT_ACTIVE`	-100	Der `SHOT_TRANSFER`-Mode oder der `SHOT_CONTAINER_MODE`-Mode ist nicht aktiviert -> Profilübertragung neu starten
`ERROR_PROFTRANS_SHOTS_COUNT_TOO_HIGH`	-101	Die Anzahl der angeforderten Profile/Container ist größer als 65535
`ERROR_PROFTRANS_MULTIPLE_SHOTS_ACTIV`	-111	Eine MultiShot Anforderung ist aktiv -> kann mit `MultiShot(0)` abgebrochen werden

2.11.4. Get profile

```
int GetProfile();
```

Übertragen eines Profiles über die serielle Schnittstelle. Diese Funktion gibt es nur für die serielle Schnittstelle.

2.11.5. Get actual profile

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

Abholen des aktuellen Profils/Containers/Video-Bildes. Dafür muss der Funktion ein Puffer übergeben werden, in den das Profil, der Container oder das Video-Bild kopiert wird. Über den Parameter `ProfileConfig` kann die für diese Abfrage gewünschte Profilkonfiguration (siehe Kapitel 3 Profil-/Container/Video-Übertragung und Kapitel 2.9.6 „Profile config“) ausgewählt werden. Ist der Container-Mode aktiviert können die Container nur mit `PROFILE` abgeholt werden.

Mit Hilfe des Pointers `pLostProfiles` kann die Anzahl der verloren gegangenen Profile/Container abgefragt werden. Dieser Wert ist größer 0, wenn zwischen zwei Aufrufen der Funktion mehrere Profile/Container empfangen wurden. Wird dieser Parameter nicht benötigt, kann alternativ auch `NULL` übergeben werden.

Mit HoldBuffersForPolling (siehe Kapitel 2.9.11 „HoldBuffersForPolling“) können in der LLT.dll mehrere Profile für das Abholen mit GetActualProfile gespeichert werden. Dadurch können die Profilverluste durch das Polling verringert werden. Es wird immer das älteste Profil aus dem internen Puffer ausgegeben. Ist der interne Puffer leer wird die Fehlermeldung ERROR_PROFTRANS_NO_NEW_PROFILE zurückgegeben.

Der Rückgabewert ist die Anzahl der in den Puffer kopierten Bytes. Ist die Anzahl kleiner oder gleich `GENERAL_FUNCTION_NOT_AVAILABLE` ist es einer der allgemeinen oder der folgenden Rückgabewerte (siehe Kapitel 2.2 „Allgemeine Rückgabewerte der Funktionen“):

Konstante für den Rückgabewert	Wert	Beschreibung
`ERROR_PROFTRANS_WRONG_PROFILE_CONFIG`	-102	Kann das geladen Profil nicht in die gewünschte Profilkonfiguration konvertieren
`ERROR_PROFTRANS_FILE_EOF`	-103	Das Dateiende beim Laden von Profilen ist erreicht
`ERROR_PROFTRANS_NO_NEW_PROFILE`	-104	Es ist seit dem letzten Aufruf von `GetActualProfile` kein neues Profil angekommen
`ERROR_PROFTRANS_BUFFER_SIZE_TOO_LOW`	-105	Die Puffergröße des übergebenen Puffers ist zu klein
`ERROR_PROFTRANS_NO_PROFILE_TRANSFER`	-106	Die Profilübertragung ist nicht gestartet und es wird keine Datei geladen

Näheres zur Profilübertragung im Kapitel 3 „Profil-/Container/Video-Übertragung“.

2.11.6. Trigger Profile

```
int TriggerProfile();
```

Auslösen einer Profilaufnahme per Befehl. Dazu ist es notwendig, dass der scanCONTROL für den externen Trigger Modus konfiguriert ist. Die Trigger Eingänge sollten nicht angeschlossen sein.

2.11.7. Trigger Container

```
int TriggerContainer();
int TriggerContainerEnable();
int TriggerContainerDisable();
```

Auslösen eines Containers per Befehl. Wenn der der Sensor im Frametrigger Modus steht (s.OpManPartB.html#...?) dann kann mit TriggerContainer der Trigger ausgelöst werden.
Wenn der Sensor nicht im Frametrigger Modus steht, dann muss der Software Containertrigger erst mit TriggerContainerEnable aktiviert werden. Der Frametrigger Eingang sollte nicht angeschlossen sein.

2.11.8. Konvertieren von Profil-Daten

```
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);
```

Konvertieren von Profilen oder partiellen Profilen in Millimeter-Werte für die Positions-Koordinate und die Abstands-Koordinate. Außerdem können alle anderen enthaltenen Informationen extrahiert werden.
Es gibt zwei Versionen der Funktion, eine für normale Profile und eine für das `PARTIAL_PROFILE`-Format.

Parameter	**Beschreibung**
`const unsigned char *pProfile`	Pointer zu dem Profil
`unsigned int nResolution`	Punkte pro Profil (64 ... 1024)
`TProfileConfig ProfileConfig`	Profil Konfiguration (`PURE_PROFILE` ... `PROFILE`)
`TScannerType ScannerType`	Messbereich und Typ des scanCONTROLs (siehe Kapitel 2.7.3 auf Seite 15)
`unsigned int nReflection`	Auszulesender Streifen (0 bis 3)
`int bConvertToMM`	Konvertieren der Positions- und Abstands-Koordinaten in Millimeter (0 = deaktiviert, 1 = aktiviert)

`TPartialProfile *pPartialProfile`	PartialProfile
`unsigned short *pWidth`	Pointer zu einem Array für die Reflektionsbreiten
`unsigned short *pMaximum`	Pointer zu einem Array für die maximalen Intensitäten
`unsigned short *pThreshold`	Pointer zu einem Array für die Thresholds
`double *pX`	Pointer zu einem Array für die Positions-Koordinaten
`double *pZ`	Pointer zu einem Array für die Abstands-Koordinaten
`unsigned int *pM0`	Pointer zu einem Array für die M0s
`unsigned int *pM1`	Pointer zu einem Array für die M1s

Die Arrays müssen mindestens die Größe der Auflösung (Punkte pro Profile) bzw. des `PointCounts` bei `PARTIAL_PROFILE` besitzen.
Werden nicht alle Informationen benötigt, können für die nicht benötigten Informationen anstatt des Arrays eine `NULL` übergeben werden.

Die Funktionen führen automatisch einen Test durch, welche Informationen in den jeweiligen Profilen vorhanden sind.
Ist der Rückgabewert größer 0 war die Funktion erfolgreich und in den einzelnen Bits des Rückgabewertes sind die gültigen Arrays codiert.

Gesetztes Bit	Konstante für das Bit	Beschreibung
8	`CONVERT_WIDTH`	Das Array für die Reflektionsbreite wurde mit Daten gefüllt
9	`CONVERT_MAXIMUM`	Das Array für die maximalen Intensitäten wurde mit Daten gefüllt
10	`CONVERT_THRESHOLD`	Das Array für die Thresholds wurde mit Daten gefüllt
11	`CONVERT_X`	Das Array für die Positions-Koordinaten wurde mit Daten gefüllt
12	`CONVERT_Z`	Das Array für die Abstands-Koordinaten wurde mit Daten gefüllt
13	`CONVERT_M0`	Das Array für die M0s wurde mit Daten gefüllt
14	`CONVERT_M1`	Das Array für die M1s wurde mit Daten gefüllt

Ist die Anzahl kleiner oder gleich `GENERAL_FUNCTION_NOT_AVAILABLE` ist es einer der allgemeinen oder der folgenden Rückgabewerte (siehe Kapitel 2.2 „Allgemeine Rückgabewerte der Funktionen“):

Konstante für den Rückgabewert	Wert	Beschreibung
`ERROR_PROFTRANS_REFLECTION_NUMBER_TOO_HIGH`	-110	Die Nummer der gewünschten Streifens ist größer 3

Für weitere Informationen siehe Kapitel 3 „Profil-/Container/Video-Übertragung“.

2.12. Is-Funktionen

Funktionen zum Abfragen von Zuständen und Verbindungen.

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

Ist der scanCONTROL über die Schnittstelle vom Typ „iInterfaceType“ verbunden? Gültige Interface-Typen sind:

Konstante für iInterfaceType	Wert	Beschreibung
INTF_TYPE_SERIAL	1	Eine Verbindung mittels der seriellen Schnittstelle
INTF_TYPE_ETHERNET	3	Eine Verbindung mittels Ethernet

```
int IsTransferingProfiles()
```

Überträgt der scanCONTROL momentan Daten?

Der Rückgabewert kann einer der folgenden Rückgabewerte sein:

Konstante für den Rückgabewert	Wert	Beschreibung
IS_FUNC_YES	1	Abgefragter Zustand oder Verbindung ist aktiv
IS_FUNC_NO	0	Abgefragter Zustand oder Verbindung ist nicht aktiv

2.13. PartialProfile-Funktionen

Das scanCONTROL bietet die Möglichkeit das übertragene Profil einzuschränken. Der Vorteil von diesem Verfahren ist eine geringere Größe der übertragenen Daten. Außerdem können damit nicht benötigte Bereiche eines Profils schon direkt im scanCONTROL weg geschnitten werden.
Dieses Feature steht bei dem scanCONTROL 28xx erst ab einer DSP-Firmwareversion von 13 zur Verfügung und kann nicht mit dem Container-Mode kombiniert werden.

Beispiele zum Übertragen von Profilen mit „Partial Profile“ sind im Kapitel 4 „LLT2800Samples“ zu finden.

2.13.1. GetPartialProfileUnitSize

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

Diese Funktion gibt die Schrittweiten zum Einstellen des partiellen Profils zurück. Ist der Parameter `pUnitSizePoint` gleich der Resolution des uneingeschränkten Profils wird dieses Feature noch nicht von der Firmware-Version Ihres scanCONTROL’s unterstützt.

2.13.2. Get/SetPartialProfile

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

Mit Hilfe dieser Funktion kann die partielle Profilübertragung des scanCONTROL’s eingestellt werden. Vorher muss die Profile Konfiguration auf `PARTIAL_PROFILE` gestellt werden (siehe Kapitel 2.9.6 „ Profile config“).

Parameter	Bedeutung
nStartPoint	Nummer des ersten zu übertragenden Punktes
nStartPointData	Erstes Byte der Punkte

nPointCount	Anzahl der zu übertragenden Punkte
nPointDataWidth	Anzahl der Bytes pro Punkt

Alle Parameter der SetPartialProfile Funktion müssen immer ein Vielfaches der jeweiligen nUnitSize der Funktion GetPartialProfileUnitSize sein.

Der Rückgabewert kann einer der allgemeinen oder der folgenden Rückgabewerte sein (siehe Kapitel 2.2 „Allgemeine Rückgabewerte der Funktionen“):

Konstante für den Rückgabewert	**Wert**	**Beschreibung**
ERROR_PARTPROFILE_NO_PART_PROF	-350	Die Profilkonfiguration ist nicht auf PARTIAL_PROFILE eingestellt -> SetProfileConfig(PARTIAL_PROFILE); aufrufen
ERROR_PARTPROFILE_TOO_MUCH_BYTES	-351	Die Anzahl der Bytes pro Punkt ist zu hoch -> nStartPointData oder nPointDataWidth ändern
ERROR_PARTPROFILE_TOO_MUCH_POINTS	-352	Die Anzahl der Punkte ist zu hoch -> nStartPoint oder nPointCount ändern
ERROR_PARTPROFILE_NO_POINT_COUNT	-353	nPointCount oder nPointDataWidth ist 0
ERROR_PARTPROFILE_NOT_MOD_UNITSIZE_POINT	-354	nStartPoint oder nPointCount sind kein vielfaches von nUnitSizePoint (siehe Kapitel GetPartialProfileUnitSize 2.13.1 „GetPartialProfileUnitSize“)
ERROR_PARTPROFILE_NOT_MOD_UNITSIZE_DATA	-355	nStartPointData oder nPointDataWidth sind kein vielfaches von nUnitSizePointData (siehe Kapitel GetPartialProfileUnitSize 2.13.1 „GetPartialProfileUnitSize“)

Der Aufbau eines Profiles ist in Kapitel 3.1 „Beschreibung der Profil-Daten“ beschrieben.
Im Normalfall werden immer nur die Position (X-Wert) und der Abstand (Z-Wert) des ersten Streifens benötigt.

2.14. Time-Funkionen

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

Diese Funktion wertet den gesamten Timestamp eines Profils aus. Sie gibt den Timestamp des Anfangs und des Endes der Belichtung und die fortlaufende Profilnummer zurück.

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

Diese Funktion wertet nur den optionalen Teil des Timestamps eines Profils aus. Sie gibt den Zählerstand des internen Zählers, die `CmmTrigger` und `CmmActive` Flags sowie den CMM-Trigger-Zähler zurück.
Das `CmmTrigger` Flag ist immer in dem Profil gesetzt in welchem auch ein CMM Triggerimpuls ausgegeben wurde. Das `CmmActive` Flag ist immer gesetzt wenn der CMM-Trigger im Allgemeinen aktiviert ist.
Für nicht benötigte Parameter kann alternativ auch `NULL` übergeben werden.

```
#include <vector>

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

//Abfragen der Aufloesung
pInterfaceLLT->GetResolution(&Resolution);

//Erstellen eines Puffers fuer die Profile
std::vector<unsigned char> ProfilData((Resolution * 4) + 16);

if(pInterfaceLLT->GetActualProfile
   (&ProfilData[0], ProfileData.size(), PURE_PROFILE, NULL) ==
  ProfileData.size())
{
  //Dekodieren des Zeitstempels und des Profil-Zaehlers
  pInterfaceLLT->Timestamp2TimeAndCount(&ProfilData[(Resolution * 4)],
        &ShutterOpen, &ShutterClose, ProfileCount);

  //Dekodieren des optionalen Zaehlers
  pInterfaceLLT->Timestamp2CmmTriggerAndInCounter(
        &ProfilData[(Resolution * 4)], &InCounter, &CmmTrigger, &CmmActive,
        &CmmCount);
}
```

2.15. Postprocessing-Funktionen

Durch das Postprocessing kann das scanCONTROL mehrere Module auf die Profile anwenden. Diese Module stehen nur in besonderen Optionen des scanCONTROL's zur Verfügung. Nähere Informationen entnehmen Sie bitte der entsprechenden Dokumentation.

2.15.1. Read/Write Postprocessing Parameter

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

Lesen oder Schreiben der aktuellen Postprocessing-Parameter. Diese Parameter können maximal 1024 DWORD's lang sein.

Der Rückgabewert kann einer der allgemeinen Rückgabewerte sein (siehe Kapitel 2.2 „Allgemeine Rückgabewerte der Funktionen").
Das Postprocessing kann über die „Processing profile data"-Eigenschaft aktiviert werden (siehe hierzu Kapitel 2.8.7„Processing profile data").

2.15.2. Postprocessing Results

Einige der Postprocessing-Module können Ergebnisse ihrer Berechnungen in das übertragene Profil integrieren. Sie werden auf die Positions- und Abstands-Koordinaten eines Streifens geschrieben.
Die folgende Funktion extrahiert aus einem gegebenen Profil die Rechenergebnisse:

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

Der Funktion wird der Profilpuffer, seine Größe, einen Puffer in den das Ergebnis kopiert wird und dessen Größe übergeben. Wird ein Profil aus einer Datei geladen muss noch ein Pointer auf die aktuelle `TPartialProfile` Struktur übergeben werden.

Der Rückgabewert ist die Anzahl der in den Puffer kopierten Bytes. Ist die Anzahl kleiner oder gleich `GENERAL_FUNCTION_NOT_AVAILABLE`, ist es einer der allgemeinen oder der folgenden Rückgabewerte (siehe Kapitel 2.2 „Allgemeine Rückgabewerte der Funktionen"):

Konstante für den Rückgabewert	Wert	Beschreibung
`ERROR_POSTPROCESSING_NO_PROF_BUFFER`	-200	Es wurde kein Profilpuffer übergeben
`ERROR_POSTPROCESSING_MOD_4`	-201	Der Parameter `nStartPointData` oder `nPointDataWidth` ist nicht durch 4 Teilbar
`ERROR_POSTPROCESSING_NO_RESULT`	-202	Kein Ergebnisblock im Profil gefunden
`ERROR_POSTPROCESSING_LOW_BUFFERSIZE`	-203	Die Puffergröße für das Ergebnis ist zu klein
`ERROR_POSTPROCESSING_WRONG_RESULT_SIZE`	-204	Die Größe des Ergebnisblocks im Profil ist nicht korrekt

2.16. File-Funktionen

Funktionen zum Laden und Speichern von Profilen oder Profilströmen. Profile können mit dem AVI-Datenformat (Standardisiertes Video-Format) gespeichert und geladen werden.

Beispiele zum Laden und Speichern von Profilen sind im Kapitel 4 „LLT2800Samples“ zu finden.

2.16.1. Speichern von Profilen

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

Speichern von Profilen. Die Profile werden dabei mit der aktuellen Profilkonfiguration gespeichert. Der Dateiname muss inklusive Endung angegeben werden. Mit dem FileType kann der gewünschte Dateityp angegeben werden.

FileType	Wert	Beschreibung
AVI	0	AVI-Datei
CSV	2	CSV- Datei (nur für Profile)
BMP	3	BMP- Datei (nur für Video Bilder)
CSV_NEG	4	CSV- Datei (nur für Profile) mit gespiegelter Z-Achse

Es wird empfohlen das AVI-Datenformat zu verwenden, da dieses Format von allen Programmen für das scanCONTROL der Micro-Epsilon gelesen werden kann.
Zum Beenden des Speicherns muss SaveProfiles(NULL,0) aufgerufen werden. Wird beim Speichern die maximale Dateigröße erreicht wird eine Fehler-Message mit dem ERROR_STOPSAVING Wert gesendet (siehe Kapitel 2.10.2 „Message“).

Der Rückgabewert kann einer der allgemeinen oder der folgenden Rückgabewerte sein (siehe Kapitel 2.2 „Allgemeine Rückgabewerte der Funktionen“):

Konstante für den Rückgabewert	Wert	Beschreibung
ERROR_LOADSAVE_WRITING_LAST _BUFFER	-50	Fehler beim deaktivieren des Speicherns, die letzten Profile der Datei können beschädigt sein oder es wurden nicht alle gespeichert
ERROR_LOADSAVE_AVI_NOT _SUPPORTED	-58	Das Betriebssystem unterstützt das AVI-Format nicht, bitte benutzen Sie Windows 2000 oder höher
ERROR_LOADSAVE_WRONG_PROFILE _CONFIG	-60	Die Profilkonfiguration oder der Filetype passt nicht zu den übertragenen Profilen/Containern/Video-Bildern
ERROR_LOADSAVE_NOT _TRANSFERING	-61	Die Profilübertragung ist nicht aktiv

2.16.2. Laden von Profilen

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

Laden von Profilen aus einer Datei. Dabei müssen fünf Pointer auf Variablen für den Dateinamen, die PartialProfile-Konfiguration, die Profilkonfiguration, den scanCONTROL-Typ (Messbereich) und die `Rearrengement`-Eigenschaft übergeben werden. Die Pointer für die Profilkonfiguration und den scanCONTROL-Typ können `NULL` sein.
Es können `*.AVI` Dateien geladen werden, welche mit der LLT.dll, dem LLT2800Demo Programm oder den scanCONTROL Programmen der Micro-Epsilon gespeichert wurden.
Nach dem Laden können mit der Funktion `GetActualProfile` die einzelnen Profile in der Datei nacheinander ausgelesen werden (siehe Kapitel 2.11.5 „Get actual profile“).
Dabei ist zu beachten, dass aus dem `PartialProfile`-Parameter die Größe für den Puffer für ein Profil/Container berechnet werden kann (`PointCount` * `PointDataWidth`).

Der Rückgabewert ist die Anzahl von Profilen/Containern in der geladenen Datei. Ist die Anzahl kleiner als `0` ist es einer der allgemeinen oder der folgenden Rückgabewerte (siehe Kapitel 2.2 „Allgemeine Rückgabewerte der Funktionen“):

Konstante für den Rückgabewert	Wert	Beschreibung
ERROR_LOADSAVE_WHILE_SAVE_PROFILE	-51	Kann Datei nicht laden, da das Speichern aktiv ist
ERROR_LOADSAVE_NO_PROFILELENGTH_POINTER	-52	Es wurde kein Pointer für die Profillänge übergeben
ERROR_LOADSAVE_NO_LOAD_PROFILE	-53	Der Filename ist `NULL`, aber es wird momentan keine Datei geladen
ERROR_LOADSAVE_STOP_ALREADY_LOAD	-54	Es wurde schon eine Datei geladen, das laden wurde gestoppt
ERROR_LOADSAVE_CANT_OPEN_FILE	-55	Kann die Datei nicht öffnen
ERROR_LOADSAVE_INVALID_FILE_HEADER	-56	Der Fileheader der zu ladenden Datei ist falsch
ERROR_LOADSAVE_AVI_NOT_SUPPORTED	-58	Das Betriebssystem unterstützt das AVI-Format nicht. Bitte benutzen Sie Windows 2000 oder höher
ERROR_LOADSAVE_NO_REARRANGEMENT_POINTER	-59	Der Pointer `pRearrengementProfile` ist `NULL`

Die Profilkonfiguration der geladenen Profile sollte immer der Profilkonfiguration der `LoadProfiles`-Funktion entsprechen. Zusätzlich kann bei einer gespeicherten Profilkonfiguration von `PROFILE` auch `QUARTER_PROFILE` und `PURE_PROFILE` oder bei `QUARTER_PROFILE` auch `PURE_PROFILE` ausgelesen werden.

Das Laden einer Datei beeinflusst nicht den Profilkonfigurationswert für das Speichern von Profilen bzw. den Callback.

Zum Beenden des Ladens muss `LoadProfiles(NULL, NULL, NULL, NULL, NULL)` aufgerufen werden.

Mit der Funktion `LoadProfilesGetPos` (siehe nächstes Kapitel) kann die aktuelle Anzahl der Profile in der Datei abgefragt werden.

2.16.3. Navigieren in einer geladenen Datei

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

Mit dieser Funktion kann die Anzahl der Profile und die aktuelle Leseposition in der geladenen Datei abgefragt werden. Wird in die Datei noch von einer anderen Instanz gespeichert kann sich die Anzahl der Profile ändern.

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

Setzen der aktuellen Leseposition in einer geladenen Datei. Um die Position auf das erste Profil zu setzen muss `nNewPosition` 0 sein.

Der Rückgabewert kann einer der allgemeinen oder der folgenden Rückgabewerte sein (siehe Kapitel 2.2 „Allgemeine Rückgabewerte der Funktionen“):

Konstante für den Rückgabewert	Wert	Beschreibung
`ERROR_LOADSAVE_FILE_POSITION_TOO_HIGH`	-57	Die gewünschte Position ist größer oder gleich der maximalen Position

2.17. Spezielle CMM-Trigger-Funktionen

Mit Hilfe der speziellen CMM-Trigger-Funktionen wird das Starten und Beenden der Profilübertragung mit aktiviertem CMM-Trigger vereinfacht. Zusätzlich können die Profile mit aktivem CMM-Trigger in eine Datei gespeichert werden. Der CMM-Trigger steht nur in bestimmten Optionen des scanCONTROL's zur Verfügung.
Zum CMM-Trigger siehe auch die Beschreibung in Kapitel 2.8.12 „CMMTrigger".

Beispiele für den CMM-Trigger sind im Kapitel 4 „LLT2800Samples" zu finden.

Starten der Profilübertragung mit CMM-Trigger:

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

Beschreibung der Parameter:

Parameter	Beschreibung
`nCmmTrigger`	Erstes Befehlswort des CMM-Triggers, welches den Divisor und die Polarität enthält
`nProfilesForerun`	Anzahl der kontinuierlich eingegangenen Profile ab der eine stabile Datenübertragung angenommen wird
`pFilename`	Dateiname für die zu speichernde Datei (muss `NULL` sein, wenn kein speichern gewünscht)
`FileType`	Datenformat der zu speichernden Datei (siehe Kapitel 2.16.1 „Speichern von Profilen")
`nTimeout`	Timeout in ms für die gesamte Funktion

Der Rückgabewert kann einer der allgemeinen oder der folgenden Rückgabewerte sein (siehe Kapitel 2.2 „Allgemeine Rückgabewerte der Funktionen"):

Konstante für den Rückgabewert	Wert	Beschreibung
`ERROR_CMMTRIGGER_NO_DIVISOR`	-400	Divisor muss > 0 sein
`ERROR_CMMTRIGGER_TIMEOUT_AFTER_TRANSFERPROFILES`	-401	Es wurden nach `TransferProfiles` keine Profile empfangen
`ERROR_CMMTRIGGER_TIMEOUT_AFTER_SETCMMTRIGGER`	-402	Nach dem setzen des CMM-Triggers sind nicht genügend Profile mit aktivem CMMTrigger angekommen

Um diese Funktion nutzen zu können, müssen Sie das zweite bis vierte Befehlswort des CMM-Triggers vor dem Aufruf dieser Funktion setzen. Das erste Befehlswort mit dem Divisor wird erst von dieser Funktion gesetzt.

Die `StartTransmissionAndCmmTrigger` Funktion startet zuerst die Profilübertragung, ohne die Profile dabei per Callback weiterzuleiten. Ist die Verbindung eingelaufen, d.h. die gewünschte Anzahl der Profile ohne einen Ausfall übertragen worden, wird der erste CMM-Trigger-Befehl mit dem Divisor an das scanCONTROL gesendet. Danach wird auf das erste Profil mit aktivem CMM-Trigger-Flag gewartet. Ab diesem Profil werden alle weiteren

Profile per Callback weitergeleitet. Zusätzlich wird, falls ein Dateiname übergeben wurde, das Speichern der Profile mit den übergebenen Dateinamen gestartet.
Tritt beim Warten auf Profile ein Timeout auf, wird die Funktion abgebrochen.

Es ist sinnvoll für `nProfilesForerun` die halbe Profilrate anzugeben (zum Beispiel 500 Profile bei 1000 Hz) und für den `nTimeout` 3000 ms.

Stoppen der Profilübertragung mit CmmTrigger:

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

Beschreibung der Parameter:

Parameter	Beschreibung
nCmmTriggerPolarity	Polarität des CMM-Triggers (0 = Low aktiv, 1 = High aktiv)
nTimeout	Timeout in ms für die gesamte Funktion

Der Rückgabewert kann einer der allgemeinen oder der folgenden Rückgabewerte sein (siehe Kapitel 2.2 „Allgemeine Rückgabewerte der Funktionen“):

Konstante für den Rückgabewert	Wert	Beschreibung
`ERROR_CMMTRIGGER_TIMEOUT_AFTER_SETCMMTRIGGER`	-402	Nach dem setzen des CMMTriggers ist kein Profile mit deaktiviertem CMM-Trigger angekommen

Die `StopTransmissionAndCmmTrigger` Funktion stoppt zuerst den CMM-Trigger, indem sie den Divisor auf 0 setzt (dabei aber die übergebene Polarität beachtet). Danach wartet sie auf das erste Profil ohne aktivem CMM-Trigger-Flag. Dieses Profil und alle folgenden werden nicht per Callback weitergeleitet, die Profilübertragung wird gestoppt und falls gespeichert wird, wird das Speichern beendet.
Tritt beim Warten auf das erste Profil ohne aktiven CMM-Trigger-Flag ein Timeout auf wird die Funktion abgebrochen.
Sinnvoll ist es für `nTimeout` eine Zeit zwischen 100 und 500 ms anzugeben.

Nach dem stoppen der Übertragung bleiben das zweite bis vierte CMM-Trigger-Befehlswort erhalten und müssen nicht neu gesetzt werden.

2.18. Fehlerwert Konvertierungs-Funktion

Übersetzen eines Fehlerwertes in einen Fehler-Text.

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

Dieser Funktion werden der zu übersetzende Fehlerwert und ein Puffer für den String übergeben.

Der Rückgabewert ist die Anzahl der in den Puffer kopierten Zeichen. Ist die Anzahl kleiner oder gleich `GENERAL_FUNCTION_NOT_AVAILABLE` ist es einer der allgemeinen oder der folgenden Rückgabewerte (siehe Kapitel 2.2 „Allgemeine Rückgabewerte der Funktionen“):

Konstante für den Rückgabewert	Wert	Beschreibung
ERROR_TRANSERRORVALUE_WRONG_ERROR_VALUE	-450	Es wurde ein falscher Fehlerwert übergeben
ERROR_TRANSERRORVALUE_BUFFER_SIZE_TO_LOW	-451	Die Größe des übergebenen Puffers ist für den String zu klein

```
char ErrorString[200];

//Umwandlung eines Fehlerwertes in einen Fehlertext
if(pInterfaceLLT->TranslateErrorValue(GENERAL_FUNCTION_NOT_AVAILABLE,
        ErrorString, sizeof(ErrorString)) > GENERAL_FUNCTION_NOT_AVAILABLE)
{
  //wenn erfolgreich -> Fehlertext ausgeben
  cout << ErrorString;
}
```

2.19. ExportLLTConfig

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

Exportieren der aktuellen Konfiguration des scanCONTROL. Diese Konfigurations-Datei enthält alle relevanten Parameter und ist vor allem für Postprocessing-Anwendungen gedacht. Das Dateiformat entspricht dem Kommunikations-Protokoll für die serielle Verbindung mit dem scanCONTROL. Die damit erzeugten Konfigurations-Dateien können ohne Änderungen mit einem Terminal Programm über die serielle Schnittstelle an das scanCONTROL gesendet werden.

Konstante für den Rückgabewert	Wert	Beschreibung
ERROR_READWRITECONFIG_CANT_CREATE_FILE	-500	Die angegebene Datei kann nicht erstellt werden.

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

Exportiert die aktuelle Konfiguration in das vom Aufrufer übergebenen Feld. Die Größe des Feldes wird im zweiten Parameter angegeben.
Wenn der Datenzeiger 0 ist, ist Rückgabewert die Anzahl der Zeichen die kopiert werden können. Ist die Anzahl kleiner oder gleich `GENERAL_FUNCTION_NOT_AVAILABLE` ist es einer der allgemeinen Rückgabewerte.

2.20. ImportLLTConfig

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

Importieren einer Konfiguration, welche mit einer der in Kapitel 2.19 beschriebenen Funktionen erzeugt wurde.

3. Profil-/Container/Video-Übertragung

Die Profile/Container/Video-Bilder werden von der LLT.dll regelmäßig vom Treiber abgeholt.
Ist der scanCONTROL über die serielle Schnittstelle verbunden, können nur `PURE_PROFILE` Profile verarbeitet werden. Diese Einschränkung resultiert aus der beschränkten Geschwindigkeit der seriellen Schnittstelle.
Ist der scanCONTROL hingegen über Ethernet verbunden kann über die Funktion `SetProfileConfig` die aktuelle Profilkonfiguration für das Speichern oder den Callback eingestellt werden (siehe Kapitel 2.9.6 „Profile config“).

Der Callback wird immer dann aufgerufen, wenn ein neues Profil, ein neuer Container oder ein neues Video-Bild empfangen wurde (siehe Kapitel 2.10.1 „Callback“). Er dient also zur Benachrichtigung. Als Parameter beinhaltet dieser Callback einen Pointer auf das soeben empfangene Profil/Container/Video-Bild. Das Profil wurde vorher schon in die aktuelle Profilkonfiguration gewandelt. Mit Hilfe dieses Pointers kann die Callback-Funktion die empfangenen Daten in einen eigenen Puffer zur Weiterverarbeitung kopieren. Wichtig ist dabei, dass die Callback-Funktion sehr kurz ist und möglichst schnell wieder beendet wird, damit der nächste Puffer vom Treiber geholt werden kann.

Für weniger zeitkritische Anwendungen bzw. Anwendungen die nicht alle Profile verarbeiten müssen ist die Funktion `GetActualProfile` gedacht (siehe Kapitel 2.11.5 „Get actual profile“). Dieser Funktion muss ein Pointer auf einen Puffer mitgegeben werden, in den von der LLT.dll das aktuelle Profil, der aktuellen Container oder das aktuelle Video-Bild kopiert wird. Dabei muss auch die gewünschte Profilkonfiguration angegeben werden, welche unabhängig von der Profilkonfiguration für das Speichern und den Callback ist.
Mit dieser Funktion können auch Profile aus einer Datei eingelesen werden.

Mit Hilfe der nachstehenden Tabelle kann die Profilgröße berechnet werden, die der Callback übergibt, oder die das Feld für die `GetActualProfile`-Funktion groß sein muss.

ProfileConfig	**Beschreibung**	**Profilgröße in Byte**
`PROFILE`	Profildaten aller vier Streifen	64 * Resolution
`QUARTER_PROFILE`	Profildaten eines Streifens	16 * Resolution + 16
`PURE_PROFILE`	Reduzierte Profildaten eines	4 * Resolution + 16

	Streifens (nur Positions- und Abstands-Werte)	
PARTIAL_PROFILE	Partielles Profil	PointCount * PointDataWidth der TPartialProfile Struktur
CONTAINER	Container-Daten	Höhe * Breite des Containers
VIDEO_IMAGE	Video-Bild des scanCONTROL's	Höhe * Breite des Video-Bildes

```
#include <vector>
//Setzen der Aufloesung
pInterfaceLLT->SetResolution(256);

//Aktivieren der Profiluebertragung und einen Moment warten (auf Profile)
pInterfaceLLT->TransferProfiles(NORMAL_TRANSFER, true);
Sleep(500);

//Erstellen eines Puffers fuer ein Profil in QUARTER_PROFILE Mode und
//abholen eines Profils
std::vector<unsigned char> vProfile(256*16+16);
if(pInterfaceLLT->GetActualProfile(&vProfile[0], ProfSize, QUARTER_PROFILE,
                                   NULL) != 256*16+16)
{
   //Das scanCONTROL sendet keine Profile
  return;
}

double XValue[256], ZValue[256];

//Konvertieren der X- und Z-Koordinaten des Profiles in Millimeter
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))
{
  //Die X- und Z-Koordinaten wurden in Millimeter umgerechnet
}
```

3.1. Beschreibung der Profil-Daten

Die Profil-Daten haben eine Breite von 64 Byte. Die Höhe der Profildaten entspricht der Anzahl der Punkte pro Profil.
Die Profil-Daten bestehen aus 4 Streifen zu je 16 Byte:

Streifen 1	Streifen 2	Streifen 3	Streifen 4

Jeder Streifen beinhaltet die Daten für ein Profil. Standardmäßig enthält der 1. Streifen das gemessene Profil und die weiteren Streifen ungültige Daten (außer dem Timestamp am Ende des 4. Streifens). Alle Streifen können in speziellen Fällen gültige Profile enthalten (z.B. wenn alle Streifen Linearisiert werden).

Weiterhin können die Ergebnisse des Post-Processings in den Streifen 1 bis 4 liegen.
Jede Zeile in einem Streifen enthält die Daten für einen Punkt und hat die folgende Struktur:

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

Position (16 Bit)	Abstand (16 Bit)

Moment 0 (32 Bit)

Moment 1 (32 Bit)

Die einzelnen Daten eines Punktes werden in folgender Tabelle erklärt:

Anzahl der Bits	Name	Beschreibung
2	Res.	Reserviert
10	Width	Breite der Reflektion in Pixel
10	Height	Maximale Intensität der Reflektion über dem Schwellwert
10	Threshold	Aktueller Schwellwert
16	Position	Positions-Koordinate (X)
16	Abstand	Abstands-Koordinate (Z)
32	Moment 0	Integrale Intensität der Reflektion
32	Moment 1	1. Moment

Die Daten liegen im **Big-Endian**-Format vor.

Die Position (X) und der Abstand (Z) werden als Integerwerte übertragen und müssen noch in Millimeter umgerechnet werden (siehe 2.11.8 Konvertieren von Profil-Daten).
Ist die Position oder der Abstand vor der Umrechnung 0 so ist dieser Punkt ungültig, das heißt das scanCONTROL konnte für diesen Punkt keinen Abstand bestimmen.

Im Normalfall wird immer nur der 1. Streifen verwendet. Nur in sehr speziellen Fällen ist es sinnvoll alle Streifen zu verarbeiten.

Im letzten Punkte des 4. Streifens befindet sich der Timestamp.

Beispiele zum Übertragen von Profilen mit unterschiedlichen Profilkonfigurationen und das Umrechnen der Profildaten in Millimeter sind im Kapitel 4 „LLT2800Samples“ zu finden.

3.1.1. Beschreibung des Datenformates PROFILE

Dieses vordefinierte Format wird Standardmäßig immer vom scanCONTROL übertragen.
Es besteht aus allen 4 Streifen. Die Größe dieses Formates beträgt: 64 * Resolution Bytes.
Dieses Datenformat verwendet das **Big-Endian**-Format.

3.1.2. Beschreibung des Datenformates QUARTER_PROFILE

Dieses vordefinierte Format kann die LLT.dll zur Datenreduktion aus dem `PROFILE` Format erzeugen. Es besteht nur aus einem Streifen. Der Timestamp befindet sich in den 16 Bytes nach dem letzten Punkt. Die Größe dieses Formates beträgt: 16 * Resolution + 16 Bytes.
Dieses Datenformat verwendet das **Big-Endian**-Format.

3.1.3. Beschreibung des Datenformates PURE_PROFILE

Dieses vordefinierte Format kann die LLT.dll zur Datenreduktion aus dem `PROFILE` Format erzeugen. Es besteht nur aus den Positions - und Abstands-Werte des jeweils ausgewählten Streifens. Sie werden hintereinander als WORDs (2 Bytes) weitergegeben. Der Timestamp befindet sich in den 16 Bytes nach dem letzten Punkt.

Die Größe dieses Formates beträgt: 4 * Resolution + 16 Bytes

Dieses Datenformat verwendet das **Little-Endian**-Format.

3.1.4. Beschreibung des Datenformates PARTIAL_PROFILE

Das PARTIAL_PROFILE wird direkt im scanCONTROL erzeugt. Die Größe und die Bedeutung der Daten des dabei übertragenen Profils hängt von den Einstellungen der Funktion `SetPartialProfile` ab (siehe Kapitel 2.13 „PartialProfile-Funktionen“ und 3.1 „Beschreibung der Profil-Daten“).
Der Timestamp befindet sich bei diesem Format immer in den letzten 16 Bytes.

In der folgenden Tabelle sind die Einstellungen des „Partial Profile“ für die Profilkonfigurationen `QUARTER_PROFILE` und `PURE_PROFILE` aufgeführt.

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

Dieses Datenformat verwendet das **Big-Endian**-Format.

3.2. Beschreibung des Datenformates CONTAINER

Im Container-Mode werden mehrere Profile zu einem Container/Bild zusammengefasst. In den Spalten stehen die einzelnen Punkte mit den ausgewählten Eigenschaften und in den Zeilen die einzelnen Profile.

Zum Beispiel:

	Z Punkt 1	**...**	**Z Punkt n**	**X Punkt 1**	**...**	**X Punkt n**
Profil 1						
Profil 2						
Profil 3						
Profil 4						

Die Auflösung eines Punktes beträgt 16 Bit. Dieser Container kann als 16 Bit Graustufen-Bitmap direkt angezeigt werden, oder es können mit Bildverarbeitungsalgorithmen Merkmale extrahiert werden. In den letzten 16 Byte jeder Zeile kann der Timestamp des jeweiligen Profils eingeblendet werden.

Die Breite des Bildes bestimmt sich aus den Einstellungen der `Rearrangement`-Eigenschaft (siehe Kapitel 2.8.13 „Rearrangement profile“). Die Höhe (= die Anzahl der Profile pro Container) kann frei gewählt werden (siehe Kapitel 2.9.8 „Profile container size“). Dieses Datenformat verwendet das **Big-Endian**-Format.

3.3. Beschreibung des Datenformates VIDEO_IMAGE

Die Video-Bilder werden als 8 Bit Graustufen Bitmap übertragen. Dabei werden nur die reinen Bilddaten übertragen. Eventuelle Header oder das Spiegeln der Zeilen müssen extern realisiert werden. Dieses Datenformat besitzt keinen Zeitstempel.

3.4. Beschreibung des Timestamps

In den letzten 16 Bytes eines Profils befindet sich der Timestamp. Die Zeit des Timestamps beginnt alle 128 Sekunden wieder bei 0.
Der Timestamp besteht aus den Zeiten des Begins und des Endes der Belichtung eines Profils und einer fortlaufenden Profilnummer.

Der Timestamp verwendet das **Big-Endian**-Format.

Flags (2 Bit)	Reserviert (6 Bit)	Profilzähler (24 Bit)
Timestamp des Belichtungszeit Begins (32 Bit)		
0x00000000 (32 Bit)		
Timestamp des Belichtungszeit Endes (32 Bit)		

Wenn der CMM-Trigger (ein optionaler Trigger) oder der interne Zähler (ein optionaler Zähler) verwendet wird hat der 3. Abschnitt des Timestamps folgende Bedeutung:

Flags (2 Bit)	Reserviert (6 Bit)	Profilzähler (24 Bit)	
Timestamp des Belichtungszeit Begins (32 Bit)			
Flankenzähler_2 (16 Bit)	CMM Trigger Flag (1 Bit)	CMM aktiv Flag (1 Bit)	CMM Triggerimpuls Zähler (Bit 14)
Timestamp des Belichtungszeit Endes (32 Bit)			

Die 32 Bit des Timestamps sind folgender maßen zusammengesetzt:

Bit Position	Beschreibung
31..25	Sekunden
24..12	Zyklus (Überlauf bei 8000)
11..00	Zyklus Offset (Überlauf bei 3072)

Zur Vereinfachung wurde in die DLL eine Konvertierungsfunktion integriert welche die einzelnen Timestamps und Zähler aus decodiert (siehe Kapitel 2.15 „Time-Funkionen“).

4. LLT2800Samples

Als Beispiel für die Integration des scanCONTROL's in eigene Projekte sind die Beispielprogramme im LLT2800Samples-Ordner gedacht. Sie stehen zur Anschauung komplett mit Quelltext zur Verfügung.

Verzeichnis	Beschreibung
LLTInfo	Einfaches Ansprechen des scanCONTROL's mit Abfragen des Namens und der Seriennummer
GetProfiles_Poll	Übertragen von Profilen zur LLT.dll und Einlesen der Profile im Polling Mode durch das Beispielprogramm
GetProfiles_Ethernet	Übertragen von Profilen zur LLT.dll und Einlesen der Profile im Polling Mode durch das Beispielprogramm über eine Ethernet-Verbindung
GetProfiles_Callback	Übertragen von Profilen zur LLT.dll und Einlesen der Profile per Callback durch das Beispielprogramm
GetProfiles_Serial	Übertragen von Profilen zur LLT.dll und Einlesen der Profile über die serielle Schnittstelle
MultiShot	Übertragen einer bestimmten Anzahl von Profilen vom scanCONTROL
PartialProfile	Übertragen von partiellen Profilen
LoadSave	Laden und Speichern von Profilen
ContainerMode	Übertragen von Profil-Containern bzw. Gray-Scale maps
VideoMode	Übertragen von Video-Bildern der Sensor-Matrix
MultiLLTs	Verwenden von mehreren scanCONTROL's in einer Anwendung
CmmTrigger	Verwenden des optionalen programmierbaren Triggers.
CSharp_GetProfiles_Poll	Übertragen von Profilen zur LLT.dll und Einlesen der Profile im Polling Mode durch das Beispielprogramm in C#
bin	Übersetzte Beispielprogramme zum Ausprobieren