Schnittstellenhandbuch LLT.dll

C++

Dokument-Version:	1.3
SDK-Version:	4.0
LLT.dll-Version:	3.9.0.2012
Datum:	21.10.2019

I. Über dieses Dokument

Dieses Dokument hat das Ziel, es dem Leser grundsätzlich zu ermöglichen einen Laserprofilscanner vom Typ scanCONTROL mittels C++ in eine eigene Software einzubinden. Basis dazu ist das Wissen um die grundlegende Verwendung der von der LLT.dll zur Verfügung gestellten Programmierschnittstelle.

Dazu werden zu Beginn, neben allgemeinen Worten zur LLT.dll selbst, die Prinzipien der Messung und die daraus resultierenden Messwerte beschrieben. Dies ist insofern nötig, um ein gewisses Verständnis für die in der Software nötigen Messdaten zu schaffen. Des Weiteren werden die verschiedenen verfügbaren Mess-/Profildatenformate und die Varianten zu deren Übertragung dargestellt. Eine Erläuterung der Einschränkungen in Hinsicht Messgeschwindigkeiten schließt den allgemeinen Teil ab.

In die eigentliche Programmierung wird mittels der Beschreibung von häufig vorkommenden Basistasks anhand von Beispielcode eingeführt. Ausführliche Programmbeispiele und die vollständige Auflistung der API sollen die eigentliche Implementierung unterstützen.

II. Versionshistorie

Version	Datum	Autor	Status
0.1	14.01.2016	DRa	Initialer Entwurf
1.0	28.03.2017	DRa, UEi	Überarbeitete Version
1.1	09.02.2018	DRa	Update
1.2a	28.02.2019	Uei	Integration sC30xx
1.2	14.06.2019	Uei	Integration sC30xx abgeschlossen
1.3	21.10.2019	THl	Integration 25xx

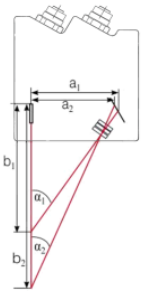
III. Inhalt

1 Einführung

1.1 Messprinzip und Messdaten

1.1.1 Prinzip der optischen Triangulation

Die scanCONTROL-Sensoren von Micro-Epsilon arbeiten, ähnlich den herkömmlichen Laserpunktsensoren, nach dem Prinzip der optischen Triangulation. Der Laserstrahl einer Laserdiode wird dabei mittels einer Spezialoptik aufgefächert und auf ein Messobjekt projiziert. Die Empfangsoptik fokussiert das diffus reflektierte Licht, welches schließlich von einem CMOS-Sensor detektiert wird. Um sicherzustellen, dass nur die Reflexion der projizierten Laserlinie ausgewertet wird, befindet sich vor dem Sensor ein Filter, der nur Licht im Wellenlängenbereich des Lasers passieren lässt.



Anhand der Position des detektierten Laserstrahls innerhalb einer Sensormatrixspalte kann nun mittels Triangulation der Abstand der Einzelmesspunkte von einer definierten Referenz im Sensor (z-Achse) bestimmt werden. In der Regel wird diese Referenz so gewählt, dass sich die Abstandswerte auf die Unterkante des Sensors beziehen. Die allgemeine Abstandsberechnung erfolgt über folgende Formel:

$$b_1 = \frac{a_1}{\tan \alpha_1}$$

Die Messauflösung in z-Richtung ist durch die Pixelanzahl der Sensormatrix in der z-Achse festgelegt. Da Reflexionen von mehreren Pixeln detektiert werden, wird zur Bestimmung des z-Wertes der Reflexionsschwerpunkt dieser Pixel verwendet (subpixelgenaue Bestimmung).

Entsprechend der Position der Messpunkte innerhalb einer Zeile der Matrix wird ein Abstandswert einem korrespondierenden Punkt auf der x-Achse zugeordnet. Die Anzahl der Pixel der Sensormatrix in x-Richtung entscheidet dann darüber, wie viele Einzelmesspunkte es gibt.

Das direkte Messergebnis ist ein zweidimensionaler Profilverlauf, welcher auf eine Maßeinheit [mm] kalibriert ist. Dadurch ist sowohl eine referenzielle, als auch eine absolute Messung möglich. Eine 3D-Messung erfolgt über eine Bewegung des Sensors oder des Messobjekts in y-Richtung. Durch gleichförmige Bewegung bei definierter Profilfrequenz oder durch Verwenden eines Encoders, der die Bewegung abbildet, kann ein Gitternetz mit äquidistant verteilten Punkten generiert werden.

1.1.2 Aufgenommene Messwerte

Die von einem scanCONTROL-Sensor standardmäßig gesendeten Daten beinhalten neben den eigentlich detektierten Abstands- und Positionswerten auch weitere Rahmendaten der Messung, wie Intensität, Reflexionsbreite, Moment 0 und Moment 1. Außerdem wird der aktuell eingestellte Schwellwert übertragen. Die Werte sollen im Folgenden erläutert werden:

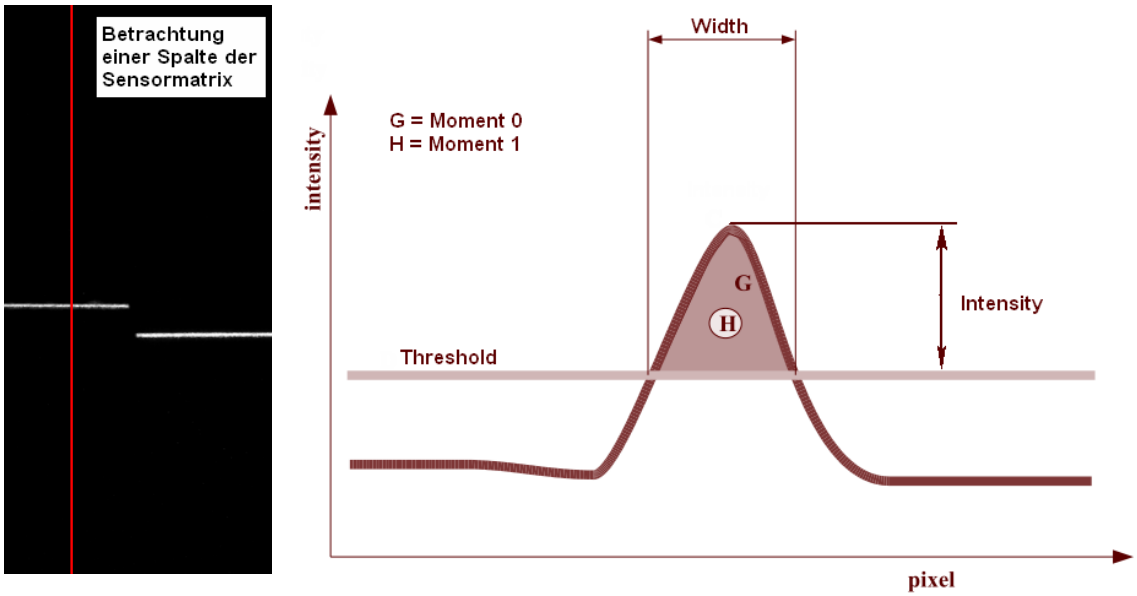


Abb. 1: Messdatenermittlung

- Abstand: Zur Ermittlung des Abstandes bzw. des z-Wertes eines Messpunktes wird der Gravitationsschwerpunkt der auf der CMOS-Sensorspalte detektierten Reflexion berechnet. Dieser wird im Sensor, anhand einer Kalibriertabelle, zu einer tatsächlichen Abstandskoordinate zurückgerechnet. Übertragen wird ein 16 Bit unsigned Integer Feld, das noch mit sensorabhängigen Skalierungsfaktoren verrechnet werden muss.

- Position: Die Position (x-Wert) korrespondiert mit der Pixelspalte des CMOS-Sensors. Pro Spalte wird ein Positionswert ermittelt. Mittels der auf dem Sensor gespeicherten Kalibriertabelle wird dieser auf die tatsächliche Position umgerechnet. Übertragen wird ebenfalls ein zu skalierendes 16 Bit unsigned Integer Feld.

- Intensität: Der übertragene Wert gibt die Differenz zwischen der maximal detektierten Intensität der aktuellen Reflexion und dem Threshold wieder. Intensität bedeutet hier, wie viel Laserlicht auf einen Pixel der Matrix gefallen ist. Voraussetzung für das Erkennen einer Reflexion ist, dass die Intensität über dem Threshold liegt. Übertragen wird ein 10 Bit unsigned Integer Feld.

- Reflexionsbreite: Die Reflexionsbreite sagt aus, über wie viele Pixel die aktuelle Reflexion zusammenhängend über dem Threshold war. Übertragen wird ein 10 Bit unsigned Integer Feld.

- Moment 0: Gibt die für die aktuelle Reflexion detektierte integrale Intensität („Fläche der Reflexion“) wieder. Das Moment ergibt sich somit aus dem Integral der über dem Schwellwert liegenden Intensität über die Reflexionsbreite; siehe Abb. 1 (G). Der Wert wird mit 32 Bit als unsigned Integer übertragen.

- Moment 1: Gibt den Schwerpunkt der Reflexion wieder, der als Basis für die Umrechnung in Abstands- und Positionswerte anhand der Kalibriertabelle verwendet wird. Der Schwerpunkt wird ebenfalls als 32 Bit unsigned Integer übertragen.

- Threshold: Der für diesen Messpunkt verwendete Schwellwert, der sich aus der eingestellten absoluten bzw. der dynamisch errechneten Schwelle und der ermittelten Fremdlichtunterdrückung zusammensetzt. Übertragen wird ein 10 Bit unsigned Integer Feld.

Es ist anzumerken, dass diese Werte bezogen auf die, im Sensor eingestellte, zu detektierende Reflexion gesendet werden. Zur Auswahl stehen die erste bzw. letzte auf der Spalte erkannte Reflexion über dem Schwellwert, die Reflexion mit der maximalen Intensität und die mit der größten integralen Intensität (siehe Abb. 2).

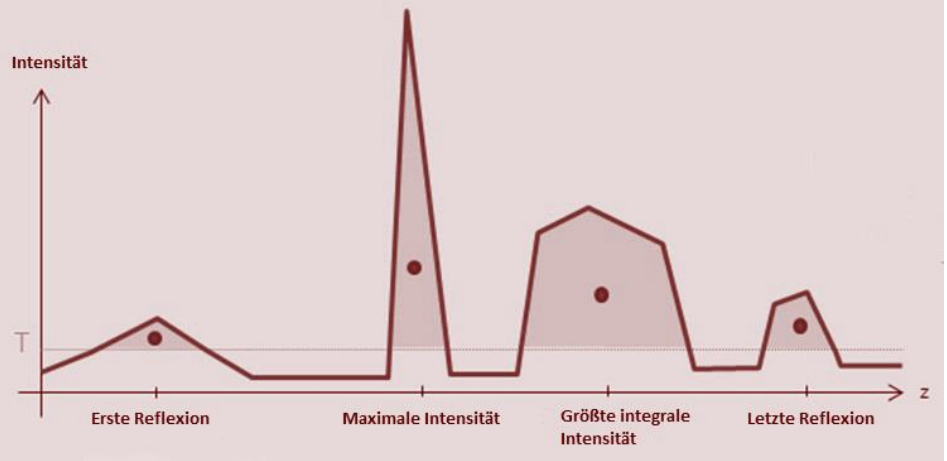


Abb. 2: Detektierbare Reflexionen

1.2 LLT.dll

Die LLT.dll ist eine *Dynamic Link Library* (DLL) zum einfachen Integrieren eines scanCONTROL Sensors 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.

Die Funktionalität umfasst die vollständige Parametrisierung und Steuerung des Sensors, die Übertragung, Speicherung und Konvertierung der Messdaten in allen verfügbaren Übertragungsmodi und die Verbindungsüberwachung zwischen PC und scanCONTROL. Außerdem ist die Möglichkeit gegeben mehrere Sensoren in eine Software einzubinden.

Um diese DLL mit möglichst vielen verschiedenen 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 unter C, Delphi oder anderen Programmiersprachen genutzt werden. Die Bedingung dafür ist die Kompatibilität der verwendeten Datentypen bzw. die Berücksichtigung der Unterschiede in der Implementierung (Bsp.: Enumerations bei Delphi). Für C++-Anwendungen gibt es eine zusätzliche Klasse mit deren Hilfe die C-Funktionen in Methoden einer Interface-Klasse gemappt werden. C# Anwendungen können realisiert werden, indem die DLL mit P/Invoke eingebunden wird.

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. Für die Implementierung mit C# steht ein zusätzliches Dokument zur Verfügung, welches dem C# Wrapper beiliegt.

1.3 Laden der DLL in C++

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.

1.3.1 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 kompiliert. 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.

1.3.2 Dynamisches Laden

Zum Laden der LLT.dll und Importieren der 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 Interface-Klasse aufgerufen werden.
Der herausragende 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 Betriebsmodi zur Profilgenerierung (nur scanCONTROL 30xx)

Die scanCONTROL 30xx-Serie bietet drei Betriebsmodi zur Profilgenerierung. Abhängig von den Anforderungen der Messaufgabe kann einer der drei verfügbaren Modi ausgewählt werden.

2.1 High Resolution Modus

Der High Resolution Modus ist der Standard-Modus zur Profilgenerierung. Der High Resolution Modus erzeugt Profildaten mit der besten Z-Linearität verglichen mit den beiden anderen Modi.

2.2 High Speed Modus

Der High Speed Modus generiert Profile mit der doppelten Profilfrequenz (verglichen mit dem High Resolution Modus bei gleicher Messfeldgröße), bei nur geringfügig geringerer Z-Linearität.

2.3 High Dynamic Range Modus (HDR)

Der High Dynamic Range Modus wird für Messobjektoberflächen verwendet, die inhomogene Oberflächeneigenschaften aufweisen (z.B. schwarz matte und helle Anteile). Hierfür werden spezielle Features der Matrix benutzt, um Profildaten auf schwierigen Oberflächen zu optimieren, ohne dass Bewegungsunschärfe entsteht.

Wie auch beim High Speed Modus ist die Linearität geringfügig schlechter als beim High Resolution Modus, da nur jede zweite Zeile verwendet wird, um Profilpunkte zu generieren.

3 Format der Messdaten

3.1 Video Mode

Im Video Mode überträgt der Sensor das vom CMOS-Sensor aufgenommene Bild als 8-Bit Graustufen Bitmap (Abb. 3). Dabei werden nur die reinen Bilddaten übertragen. Eventuelle Header oder das Spiegeln der Zeilen müssen extern realisiert werden. Dieses Datenformat besitzt keinen Zeitstempel. Die Messfrequenz sollte in diesem Modus 25 Hz nicht übersteigen. Zu beachten ist außerdem, dass bei Ethernet-Scannern der 29xx-Serie eine Gigabit-Ethernet-Verbindung nötig ist, um die Bilddaten mit der empfohlenen Frequenz von 25 Hz zu übertragen. Der Scanner der 29xx-Serie benötigt z.B. eine Bandbreite von ca. 262 Mbit/s bei 25 Hz (25 Hz * 1024 * 1280 * 8 Bit).

Abb. 3: Beispiel Video Mode

Scanner der Serie scanCONTROL 30xx besitzen einen hochauflösenden Video Mode der eine Bandbreite von 445 Mbit/s benötigt (25 Hz * 1088 * 2048 * 8 bit) und einen niedrig auflösenden Video Mode der nur 28 Mbit/s benötigt (25 Hz * 272 * 512 * 8 bit).

3.2 Einzelprofilübertragung

3.2.1 Profildatenformat allgemein

Im Einzelprofilmodus wird standardmäßig pro Messung ein Datenfeld übertragen, das pro Messpunkt 64 Byte breit ist. Die Höhe des Datenfeldes wird von der Anzahl der Punkte pro Profil festgelegt. Die 64 Byte unterteilen sich in vier Streifen mit jeweils 16 Byte Breite. Jeder Streifen kann ein komplettes Profil enthalten; im Normalfall enthält nur der erste Streifen gültige Profildaten - sind Mehrfachreflexionen vorhanden, können aber auch die anderen Streifen Profildaten enthalten. Die letzten 16 Byte des Datenfeldes enthalten den Timestamp, was in der Standardeinstellung (*Full Set*) dem letzten Punkt des vierten Streifens entspricht.

Pro Streifen werden für jeden Messpunkt alle Informationen übertragen, die in den vorherigen Abschnitten beschrieben wurden. Diese sind jeweils in folgender Struktur angeordnet:

0..7	8..15	16..23	24..31

Res. (2 Bit)	Reflexionsbreite (10 Bit)	Max. Intensität (10 Bit)	Threshold (10 Bit)
Position (16 Bit)		Abstand (16 Bit)	
Moment 0 (32 Bit)			
Moment 1 (32 Bit)			

Die byteweise Daten-Formatierung der einzelnen Messwerte ist mit Big-Endian ausgeführt. Außerdem müssen, wie bereits angedeutet, die Positions- und Abstandsdaten (X/Z) noch mittels Skalierungsfaktoren umgerechnet werden. Diese sind für jeden Messbereich verschieden. Abb. 4 zeigt schematisch die Gesamtstruktur einer Übertragung.

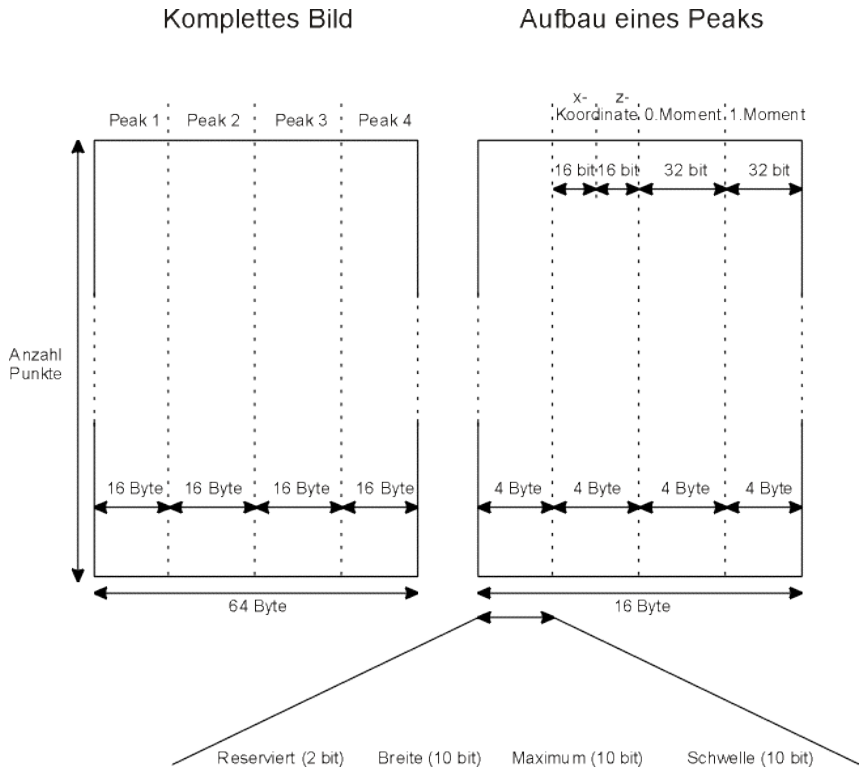


Abb. 4: Aufbau Profilübertragung

3.2.2 Timestamp-Informationen

Der in den letzten 16 Byte übertragene Timestamp beinhaltet folgende Rahmendaten eines gemessenen Profils (Datenformat: unsigned Integer; Big-Endian):

- Profilzähler: Inkrementeller Zähler mit dem sich Profile identifizieren lassen; wird bei jedem neuen Profil um eins erhöht. Für das Feld sind 24 Bit reserviert – es läuft damit nach 16777215 Profilen über.

- Startzeit Belichtung: Beinhaltet den absoluten Zeitpunkt bei dem die Belichtung gestartet wurde. Die interne Uhr hat dabei eine Periode von 128 Sekunden. Das 32-Bit breite Feld besteht aus einem Sekundenzähler, einem Zykluszähler und dem Zyklusoffset. Aus diesen Werten lässt sich der eigentliche Verschlussöffnungszeitpunkt berechnen.

- Flankenzähler: Hier wird je nach Scannereinstellung entweder die zweifache Anzahl der erkannten Encoderflanken oder der Status der Digitaleingänge übertragen. Das Feld ist 16-Bit groß.

- Endzeit Belichtung: Beinhaltet den absoluten Zeitpunkt bei dem die Belichtung beendet wurde. Ansonsten wie bei *Startzeit Belichtung* beschrieben.

Struktur der Timestamp-Daten ist wie folgt:

0..7	8..15	16..23	24..31
Flags (2 Bit)	Reserviert (6 Bit)	Profilzähler (24 Bit)	
Startzeit Belichtung (32 Bit)			
Flankenzähler bzw. DigIn (16 Bit)		Reserviert (16 Bit)	
Endzeit Belichtung (32 Bit)			

Die beiden Zeitstempel für die Belichtung setzen sich folgendermaßen zusammen:

Sekundenzähler (7 Bit)	Zykluszähler (13 Bit)	Zyklusoffset (12 Bit)

Der eigentliche Zeitstempel folgt dann aus folgender Berechnungsvorschrift:

$$\text{Zeitstempel} = \text{Sekundenzähler} + \frac{\text{Zykluszähler}}{8000} + \frac{\text{Zyklusoffset}}{8000*3072}$$

(Bemerkung: Der Zykluszähler läuft bei 8000 und der Zyklusoffset bei 3072 über!)

3.2.3 CMM-Timestamp

Ist die Triggerung für eine *Coordinate Measuring Machine* (CMM; Koordinatenmessmaschine) aktiviert, ändert sich das Format des Timestamps. Statt dem Encoder-Flankenzähler und der reservierten 16-Bit werden folgende CMM-spezifische Informationen übertragen:

CMM-Flankenzähler (16 Bit)	CMM Trigger Flag (1 Bit)	CMM aktiv Flag (1 Bit)	CMM Triggerimpulszähler (14 Bit)

3.2.4 Alle Messdaten (Full Set, PROFILE)

Standardmäßig werden vom Sensor alle Messdaten wie in **1.1.2** beschrieben übertragen. Mit dem vordefinierten Format *Full Set* werden alle Daten aus der Übertragung extrahiert. Die Datenmenge pro Messung ist hier mit 64 Byte mal der Anzahl der Punkte pro Profil anzusetzen. Im Rahmen der DLL wird diese Konfiguration auch als Profilkonfiguration *PROFILE* bezeichnet. Ausgabeformat ist Big-Endian.

3.2.5 Ein Streifen (QUARTER_PROFILE)

Die Profilkonfiguration *QUARTER_PROFILE* ermöglicht es, nur einen Streifen aus der Übertragung zu extrahieren. Dies hat zur Folge, dass eine kleinere Datenmenge verarbeitet werden muss. Die Timestamp-Informationen werden an die Profildaten angehängt. Die Daten pro Messung beschränken sich daher hier auf 16 Byte mal der Punkte pro Profil plus 16 Byte Timestamp. Es ist zu beachten, dass die Profilkonfiguration per se nicht die übertragene Datenmenge reduziert, sondern nur der Teil des Puffers ausgewertet wird, der dem gewünschten Streifen entspricht. Es muss daher weiterhin die volle Datenmenge übertragen werden. Ausgabeformat ist Big-Endian.

3.2.6 X/Z-Daten (PURE_PROFILE)

Die Profilkonfiguration *PURE_PROFILE* extrahiert nur die Abstands- und Positionswerte (X/Z-Daten) aus dem aktuell ausgewählten Streifen. Ansonsten verhält sie sich wie die *QUARTER_PROFILE* Konfiguration, d.h. die tatsächlich übertragene Datenmenge wird nicht

reduziert. Die ausgewertete Datenmenge reduziert sich auf 4 Byte mal der Punkte pro Profil plus 16 Byte Timestamp. Zu beachten ist, dass hier im Little-Endian-Format übertragen wird.

3.2.7 Partielles Profil (PARTIAL_PROFILE)

Ein partielles Profil (*PARTIAL_PROFILE*) wird direkt im Scanner erzeugt und kann daher die zu übertragende Datenmenge stark reduzieren. Auch das Processing der Daten im Scanner wird beschleunigt, was bei hohen Frequenzen wichtig werden kann. Die Größe des Profils kann mittels der folgenden vier Parameter definiert werden:

1. *StartPoint*: Erster Messpunkt der im Profil enthalten sein soll
2. *StartPointData*: Offset ab welchem Byte die Daten eines Punktes im Profil enthalten sein sollen
3. *PointCount*: Anzahl der Messpunkte ab dem *StartPoint*, die im Profil enthalten sein sollen
4. *PointDataWidth*: Anzahl an Bytes ab dem *StartPointData*-Offsets, die im Profil enthalten sein sollen

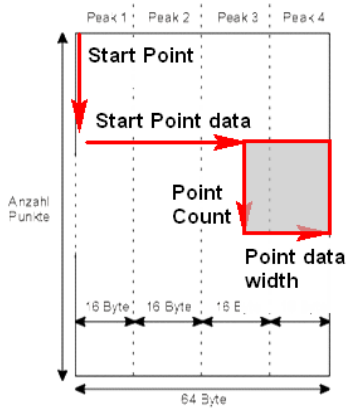


Abb. 5: Illustration Partial Profile

Alle Angaben haben als Bezugspunkt den linken oberen Punkt der Profildaten und der Basisindex ist jeweils 0. Die Angaben müssen durch die auf dem Sensor festgelegte Unitsize teilbar sein (meist 4). Am Ende des partiellen Profils befindet sich wieder der Timestamp, jedoch wird er nicht angehängt, sondern überschreibt die letzten 16 Bytes. Die übertragene Datenmenge reduziert sich somit auf *PointDataWidth* Bytes mal *PointCount*. Ausgabeformat ist Big-Endian.

3.3 Container Mode

Prinzipiell erlaubt der Container Mode eine Zusammenfassung von Daten aus mehreren Profilen in einen großen Übertragungscontainer. Dies hat unter anderem den Vorteil, dass die nötige Reaktionszeiten der Software und der Daten-Overhead reduziert werden kann.

3.3.1 Standard Container Mode

Standardmäßig können im Container Mode mehrere komplette Profile in einen logischen Übertragungscontainer zusammengefasst werden. Dabei sammelt der Sensor so lange Profile, bis die gewünschte Anzahl erreicht ist und überträgt diese als Gesamtpaket. Die maximale Containergröße hängt vom Sensor ab (128 Mbyte bzw. 4096 Profile). Wie erwähnt, ist hier der Hauptvorteil, dass die korrespondierende Software in längeren Reaktionsintervallen arbeiten kann.

3.3.2 Rearranged Container Mode (Transponierter Container Mode)

Mit der sog. *Rearrangement*-Funktion können nur die Daten in den Container geschrieben werden, die tatsächlich nötig sind. Der Anwender kann dabei frei bestimmen, von welchem Streifen er welche Messwerte übertragen will. Diese Werte werden dann pro Profil hintereinander im Container abgespeichert. Für den Timestamp kann man ein zusätzliches Feld definieren, falls gewünscht. Die gewählten Werte werden dann für eine gewünschte Anzahl von Profilen gesammelt und dann übertragen.

Ein häufig verwendeter Sonderfall ist der transponierte Container Mode, bei dem nur die Abstandsdaten (und optional Positionsdaten) extrahiert werden. Diese sind dann so angeordnet, dass sie ein 16-Bit Graustufen-Bitmap ergeben, was sich anschließend mittels Bildverarbeitungsalgorithmen analysieren lässt. Die Breite des Bitmaps wird hierbei von der Anzahl der Punkte pro Profil und die Höhe von der Anzahl der Profile im Container festgelegt. Standard-Datenformat ist Big-Endian, kann aber in Little-Endian geändert werden.

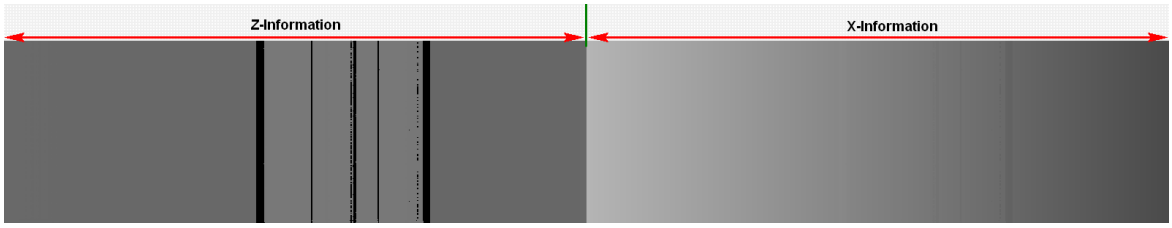


Abb. 6: Beispielbild transponierter Container Mode

4 Datenübertragung vom scanCONTROL Sensor

4.1 Datenübertragung

Die Datenübertragung wird softwareseitig gestartet und beendet. Ist die Übertragung aktiv, werden abhängig von der Profilfrequenz Daten in einen Empfangspuffer geschrieben. Konfiguriert werden kann die Übertragung entweder für eine kontinuierliche Datenextraktion aus dem Puffer (*NORMAL_TRANSFER*) oder für die Extraktion einer definierten Anzahl an Profilen (*SHOT_TRANSFER*).

4.2 Pollen von Messdaten

Für weniger zeitkritische Anwendungen bzw. Anwendungen, die nicht alle empfangenen Profile oder Videobilder verarbeiten müssen, gibt es die Möglichkeit Daten aktiv aus dem Empfangspuffer anzufordern. Bei jedem Poll wird das zuletzt empfangene Profil/Bild ausgelesen und in einen von der Anwendersoftware bereitgestellten Datenpuffer kopiert, der zur Weiterverarbeitung verwendet werden kann. Ist seit dem letzten Poll noch kein neues Profil angekommen, wird die Anwendung informiert.

4.3 Nutzen von Callbacks

Der Callback wird immer dann aufgerufen, wenn ein neues Profil oder ein neuer Container empfangen wurde. Bei vollständiger Abarbeitung der Callback-Routine ist ein Event zu setzen. Als Parameter beinhaltet dieser Callback einen Zeiger auf das soeben empfangene Profil bzw. den empfangenen Container. Das Profil wurde vorher schon in die aktuelle Profilkonfiguration gewandelt. Mit Hilfe dieses Zeigers 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.

5 Messgeschwindigkeit

Die maximal mögliche Messgeschwindigkeit hängt von mehreren Parametern ab und unterscheidet sich je nach Sensortyp. Zusätzlich hat die gegebene Netzwerkinfrastruktur eine einschränkende Rolle. Eine Überschreitung dieser Maximalfrequenz führt zu verlorenen und/oder korrupten Daten und sollte komplett vermieden werden.

Um von vorneherein Performanceprobleme zu vermeiden, sollte die komplette Netzwerkinfrastruktur zwischen Sensor und PC Gigabit-Ethernet tauglich sein. Speziell falls Sensoren der 29xx-Serie oder der 30xx-Serie einsetzt werden, kommen 100 Mbit/s-Netzwerke schnell an die physikalischen Grenzen. Außerdem sollte der Rechner, auf dem die Software ausgeführt wird, mit ausreichend performanter Hardware ausgestattet sein, da v.a. bei mehreren Sensoren eine große Datenmenge abzuarbeiten ist.

Häufigster Flaschenhals, nach der Netzwerkumgebung, ist das eingesetzte Messfeld. Das Messfeld beschreibt, welcher Teil der Sensormatrix tatsächlich ausgelesen wird. Je weniger Fläche der Sensormatrix ausgelesen wird, desto größer ist die Maximalfrequenz. Eine detaillierte Auflistung der je nach Messfeld möglichen Frequenzen, ist in der (der Sensordokumentation beiliegenden) *Quick Reference* gegeben. Grundsätzlich sollte bei hochfrequenten Messungen nur der für die Messung relevante Teil der Matrix ausgelesen werden.

Weiterhin beschränkt die Länge der Belichtungszeit die Messfrequenz. Die Maximalfrequenz ergibt sich hier aus dem Reziprokwert der Belichtungsdauer. Sind extensive Post-Processing Operationen auf dem Sensor konfiguriert worden (SMART- oder GAP-Sensoren), kann auch hier ein Flaschenhals entstehen. Die mit der aktuellen Post-Processing-Konfiguration mögliche Maximalfrequenz kann den *scanCONTROL Configuration Tools* entnommen werden. Bei höheren Frequenzen kann auch bei Sensoren ohne aufgespielten Processing die Berechnungszeit nicht ausreichen. Dies kann vermieden werden, indem man die auszuwertenden Punkte dem Messfeld anpasst und/oder nur einen Streifen bzw. die X/Z-Werte überträgt (via partiellem Profil). Vor allem bei der jeweiligen Maximalgeschwindigkeit ist auf die Minimalkonfiguration zurückzufallen. Dies hat keine Auswirkungen auf die Qualität der Messung, da alle weiteren Punkte außerhalb des Messfeldes liegen und nur invalide Werte liefern würden.

6 Typische Code-Beispiele mit Verweise auf das SDK

Im folgenden Abschnitt werden Code-Minimalbeispiele für verschiedene Einbindungsschritte gezeigt. Komplettbeispiele mit Fehlerbehandlung etc. sind im Projektordner des SDKs zu finden. Bis auf Beispiele 5.1 und 5.13 ist vorausgesetzt, dass die Verbindung mit dem Sensor hergestellt ist. Bei einigen Beispielen werden Register beschrieben (*SetFeature(…)*). Die Registeradressen sind in der SDK auf Makros abgebildet (z.B. `FEATURE_FUNCTION_EXPOSURE_TIME` für die Belichtungszeit). Die detaillierten Registerbeschreibungen sind im Operation Manual Part B zu finden (Siehe Abschnitt 8.3); diese ist der Scannerdokumentation beiliegend.

6.1 Verbindung mit Sensor herstellen

Dieses Beispiel zeigt, wie ein Sensor gefunden und eine Verbindung hergestellt werden kann.

```
std::vector<unsigned int> Interfaces(5);
static CInterfaceLLT* pLLT = nullptr;
bool LoadError;

// Erzeugen eines Handles für einen Ethernet Scanner
pLLT = new CInterfaceLLT("LLT.dll", &LoadError);

// Zuweisen einer Schnittstelle
pLLT->CreateLLTDevice(INTF_TYPE_ETHERNET);

// Suchen nach Scannern am Interface
pLLT->GetDeviceInterfacesFast(&Interfaces[0], (unsigned int)Interfaces.size());

// Zuordnen des ersten gefundenen Interfaces zum Handle
pLLT->SetDeviceInterface(Interfaces[0], 0);

// Verbindung herstellen
pLLT->Connect();
```

Siehe API: CreateLLTDevice(), GetDeviceInterfaces(), GetDeviceInterfacesFast(), SetDeviceInterface(), Connect()

6.2 Profilfrequenz und Belichtungszeit setzen (nur scanCONTROL 30xx)

Dieses Beispiel zeigt die Vorgehensweise zum Ändern der Profilfrequenz und der Belichtungszeit für den scanCONTROL 30xx. Die übergebenen Werte beschreiben die Zeit in 1 µs-Schritten. Die Profilfrequenz kann nicht direkt gesetzt werden. Sie setzt sich aus der Belichtungszeit (*ExposureTime*) und der Leerlaufzeit (*IdleTime*) zusammen. Sie berechnet sich aus:

$$Profilfrequenz = \frac{1}{(ExposureTime + IdleTime)}$$

```
unsigned int ExposureTime      = 1005;
unsigned int IdleTime          = 8995;

// Setzen der Belichtungszeit
pLLT->SetFeature(FEATURE_FUNCTION_EXPOSURE_TIME,
    (((ExposureTime % 10) << 12) & 0xF000) +
    ((ExposureTime / 10) & 0xFFF));
```

```
// Setzen der Leerlaufzeit
pLLT->SetFeature(FEATURE_FUNCTION_IDLE_TIME,
    (((IdleTime % 10) << 12) & 0xF000) +
    ((IdleTime / 10) & 0xFFF));
```

Siehe API: SetFeature(), FEATURE_FUNCTION_EXPOSURE_TIME, FEATURE_FUNCTION_IDLE_TIME
Siehe Quickreference: *OpManPartB.html#exposuretime, OpManPartB.html#idletime*

Im Beispielcode wird also die Belichtungszeit auf 1.005 ms und die Profilfrequenz auf 100 Hz festgelegt. Siehe SDK-Beispiele (sC30xx_HighSpeed).

6.3 Profilfrequenz und Belichtungszeit setzen (alle scanCONTROL Typen)

Dieses Beispiel zeigt die Vorgehensweise zum Ändern der Profilfrequenz und der Belichtungszeit. Die übergebenen Werte beschreiben die Zeit in 10 µs-Schritten. Die Profilfrequenz kann nicht direkt gesetzt werden. Sie setzt sich aus der Belichtungszeit (*ExposureTime*) und der Leerlaufzeit (*IdleTime*) zusammen. Sie berechnet sich aus:

$$Profilfrequenz = \frac{1}{(ExposureTime + IdleTime) * 10\ \mu s}$$

```
unsigned int ExposureTime      = 100;
unsigned int IdleTime          = 900;

// Setzen der Belichtungszeit auf 1 ms (100*10 us)
pLLT->SetFeature(FEATURE_FUNCTION_EXPOSURE_TIME, ExposureTime);

// Setzen der Leerlaufzeit auf 9 ms (900*10 us)
pLLT->SetFeature(FEATURE_FUNCTION_IDLE_TIME, IdleTime);
```

Siehe API: SetFeature(), FEATURE_FUNCTION_EXPOSURE_TIME, FEATURE_FUNCTION_IDLE_TIME
Siehe Quickreference: *OpManPartB.html#exposuretime, OpManPartB.html#idletime*

Im Beispielcode wird also die Belichtungszeit auf 1 ms und die Profilfrequenz auf 100 Hz festgelegt.

6.4 Pollen von Messwerten

Dieses Beispiel zeigt die Profildatenabholung über aktives Pollen. Dazu wird mittels der Funktion *GetActualProfile()* das zuletzt empfangene Profil, das sich im Empfangspuffer befindet, in einen Puffer der Anwendung zur Weiterverarbeitung kopiert. Anschließend wird aus den Pufferdaten die eigentliche X/Z-Information extrahiert (*ConvertProfile2Values()*). Diese Funktion rechnet schon die nötigen Skalierungsfaktoren für den Scannertyp ein; die Ausgabewerte sind Positions- und Abstandswerte in Millimeter.

```
unsigned int Resolution = resolutions[0];
TScannerType scanCONTROLType;

[…] // Connect

// Festlegen von Puffer für ein vollständiges Profil und X/Z-Werte
std::vector<unsigned char>ProfileBuffer(Resolution * 64);
std::vector<double>ValueX(Resolution);
std::vector<double>ValueZ(Resolution);
```

```
// Scanner-Typ abfragen
pLLT->GetLLTType(&scanCONTROLType);

// Lost profiles Zähler
unsigned int LostProfiles = 0;

// Starten der kontinuierlichen Profilübertragung
pLLT->TransferProfiles(NORMAL_TRANSFER, true);

// Pollen eines Profiles und Abspeichern in Puffer
// Anm.: Falls noch kein neues Profil seit dem letzten Aufruf angekommen ist
// gibt die Funktion -104 zurück. Ggfls. in einer Schleife abfragen.
pLLT->GetActualProfile(&ProfileBuffer[0], (unsigned int)ProfileBuffer.size(),
                                                    PROFILE, &LostProfiles));

// Konvertierung von Pufferdaten zu X/Z-Werten des Profils
pLLT->ConvertProfile2Values(&ProfileBuffer[0], ProfileBuffer.size(), Resolution,
    PROFILE, scanCONTROLType, 0, 1, NULL, NULL, NULL, ValueX, ValueZ, NULL, NULL);

// Beenden der kontinuierlichen Profilübertragung
pLLT->TransferProfiles(NORMAL_TRANSFER, false);
```

Siehe API: GetLLTType(), TransferProfiles(), GetActualProfile(), ConvertProfiles2Values()

6.5 Auslesen via Callback

Dieses Beispiel zeigt die Profildatenabholung mittels Callback. Dazu wird ein Callback registriert, der bei einem neu angekommenen Profil den Puffer kopiert und anschließend ein Event signalisiert. Nach Empfangen des Profils wird die Übertragung beendet.

```
unsigned int Resolution;
unsigned int ProfileBufferSize;
TScannerType scanCONTROLType;

// Callback handle
HANDLE hProfileEvent = CreateEvent(NULL, true, false, "ProfileEvent");

// Puffer reservieren
std::vector<double> ValueX(Resolution);
std::vector<double> ValueZ(Resolution);
std::vector<unsigned char>ProfileBuffer(Resolution * 64);

// Registrieren der Callback-Funktion
pLLT->RegisterCallback(STD_CALL, (void*)NewProfile, pLLT)

// Starten der kontinuierlichen Profilübertragung
pLLT->TransferProfiles(NORMAL_TRANSFER, true);

// Warte auf Event mit Timeout 1 Sekunde
if(WaitForSingleObject(hProfileEvent, 1000) != WAIT_OBJECT_0)
{
    // Timeout behandeln
}

// Beenden der kontinuierlichen Profilübertragung
pLLT->TransferProfiles(NORMAL_TRANSFER, false);
```

```
// Callback-Funktion (kopiert ein Profil in den Puffer und signalisiert danach)
void __stdcall NewProfile(const unsigned char* pucData, unsigned int uiSize,
                                                                    void* pUserData)
{
   memcpy(&ProfileBuffer[0], pucData, uiSize);
   SetEvent(hProfileEvent);
}
```

Siehe API: RegisterCallback(), TransferProfiles()

Siehe SDK-Beispiele (GetProfiles_Callback).

6.6 Profilfilter setzen

Dieses Beispiel zeigt das Setzen von Resampling-, Median- und Average-Filter.

```
// Setze Average-Filter auf 7 Taps
unsigned int ProfileFilter = FILTER_AVG_7;
// Setze Median-Filter auf 5 Taps
ProfileFilter |= FILTER_MEDIAN_5;
// Setze Resampling (Inter/Extrapolation von invaliden Punkten; Alle Infos; huge)
ProfileFilter |= FILTER_RESAMPLE_EXTRAPOLATE_POINTS |
                        FILTER_RESAMPLE_ALL_INFO | FILTER_RESAMPLE_HUGE;
// Setzen der eingestellten Filter
pLLT->SetFeature(FEATURE_FUNCTION_PROFILE_FILTER, ProfileFilter);
```

Siehe API: SetFeature(), FEATURE_FUNCTION_PROFILE_FILTER
Siehe Operational Manual Part B: *OpManPartB.html#profilefilter*

6.7 Encoder

Dieses Beispiel zeigt das Aktivieren der Encoder-Triggerung über digitale Eingänge.

```
unsigned int Encoder = 0;

// Setze Trigger input auf encoder
unsigned int Trigger = TRIG_MODE_ENCODER;
// Setze digitale Eingänge als Trigger input und aktiviere ext. Triggerung
Trigger |= TRIG_INPUT_DIGIN | TRIG_EXT_ACTIVE;
// Setzen der Triggereinstellungen
pLLT->SetFeature(FEATURE_FUNCTION_TRIGGER, Trigger);

// Setzen des Multifunktionsports auf bidirektionalen 24V HTL-Encoderbetrieb
unsigned int MultiPort = MULTI_DIGIN_ENC_INDEX | MULTI_LEVEL_24V
                                                    | MULTI_ENCODER_BIDIRECT;
pLLT->SetFeature(FEATURE_FUNCTION_DIGITAL_IO, MultiPort);

// Lese Maintenance-Register und aktivere Encoder
pLLT->GetFeature(FEATURE_FUNCTION_MAINTENANCE, &Encoder);
Encoder |= MAINTENANCE_ENCODER_ACTIVE;
pLLT->SetFeature(FEATURE_FUNCTION_MAINTENANCE, Encoder);
```

Siehe API: SetFeature() ,GetFeature(), FEATURE_FUNCTION_TRIGGER, FEATURE_FUNCTION_MAINTENANCE, FEATURE_FUNCTION_DIGITAL_IO
Siehe Operational Manual Part B: *OpManPartB.html#trigger, OpManPartB.html#maintenance, OpManPartB.html #ioconfig*

6.8 Externe Triggerung

Dieses Beispiel zeigt das Aktivieren der externen Triggerung über digitale Eingänge.

```
// Setze Trigger input auf pos. pulse mode
unsigned int Trigger = TRIG_MODE_PULSE | TRIG_POLARITY_HIGH;
// Setze digitale Eingänge als Trigger input und aktiviere ext. Triggerung
Trigger |= TRIG_INPUT_DIGIN | TRIG_EXT_ACTIVE;
// Setzen der Triggereinstellungen
pLLT->SetFeature(FEATURE_FUNCTION_TRIGGER, Trigger);

// Setzen des Multifunktionsports auf 5V TTL-DigIn-Trigger
unsigned int MultiPort = MULTI_DIGIN_TRIG_ONLY | MULTI_LEVEL_5V;
pLLT->SetFeature(FEATURE_FUNCTION_DIGITAL_IO, MultiPort);
```

Siehe API: SetFeature(), FEATURE_FUNCTION_TRIGGER, FEATURE_FUNCTION_DIGITAL_IO
Siehe Operational Manual Part B: *OpManPartB.html#trigger, OpManPartB.html#ioconfig*

6.9 Software-Profil-Trigger

Dieses Beispiel zeigt die externe Profil-Triggerung über den Software-Profil-Trigger.

```
// Setzen der Triggereinstellungen
pLLT->SetFeature(FEATURE_FUNCTION_TRIGGER, TRIG_EXT_ACTIVE);

// Software-Triggerung; Löst Aufnahme eines Profils aus
pLLT->TriggerProfile();
```

Siehe API: SetFeature(), TriggerProfile(), FEATURE_FUNCTION_TRIGGER
Siehe Operational Manual Part B: *OpManPartB.html#trigger*

Siehe SDK-Beispiele (TriggerProfile).

6.10 Software-Container-Trigger

Dieses Beispiel zeigt die externe Container-Triggerung über den Software-Container-Trigger. Um diese Funktion zu nutzen, muss der Sensor entweder im Frametrigger-Modus sein (siehe Digital-IO Konfiguration), oder der Container Trigger muss explizit aktiviert werden (TriggerContainerEnable() / TriggerContainerDisable()).

```
// Aktivieren des Container-Triggers, falls notwendig
pLLT->TriggerContainerEnable();

// Starten der Container-Übertragung
pLLT->TransferProfiles(NORMAL_CONTAINER_MODE, true);

// Einen Container triggern
pLLT->TriggerContainer();

[…] // Auf Container warten und Container aus dem Puffer holen

// Container-Übertragung beenden
pLLT->TransferProfiles(NORMAL_CONTAINER_MODE, false);

// Deaktivieren des Container-Triggers, falls notwendig
pLLT->TriggerContainerDisable();
```

Siehe API: SetFeature(), TriggerContainer(), FEATURE_FUNCTION_TRIGGER
Siehe Operational Manual Part B: *OpManPartB.html#trigger*

Voraussetzung: Firmware Version v46 oder neuer.
Siehe SDK-Beispiele (TriggerContainer).

6.11 Peak-Filter setzen

Dieses Beispiel zeigt, wie die sog. Peak-Filter des Scanners gesetzt werden können. Diese ermöglichen es, Punkte außerhalb eines gewissen Intensitäts- und/oder Reflexionsbreitenbereich auszusortieren. Um die Einstellungen zu aktivieren, muss eine 0 in das FEATURE_FUNCTION_EXTRA_PARAMETER-Register geschrieben werden.

```
// Setze die gewünschten Peak-Werte
unsigned short min_width       = 2;   // Werte <2 erhöhen Rauschen immens
unsigned short max_width       = 1023;
unsigned short min_intensity   = 0;
unsigned short max_intensity   = 1023;

pLLT->SetFeature(FEATURE_FUNCTION_PEAKFILTER_WIDTH,
                                          (min_width << 16) + max_width);
pLLT->SetFeature(FEATURE_FUNCTION_PEAKFILTER_HEIGHT,
                                          (min_intensity << 16) + max_intensity);

// Abschluss des Kommandos
pLLT->SetFeature(FEATURE_FUNCTION_EXTRA_PARAMETER, 0);
```

Siehe API: SetFeature(), FEATURE_FUNCTION_PEAKFILTER, FEATURE_FUNCTION_EXTRA_PARAMETER
Siehe Operational Manual Part B: *OpManPartB.html#extraparameter*

Achtung: für die scanCONTROL Serien 27xx, 26xx, 29xx (Firmware version < v43) muss der Peakfilter über das EXTRA_PARAMETER-Register gesetzt werden. Siehe SDK-Beispiele (advanced/LLTPeakFilter).

6.12 Berechnung der Auswahlbereiche auf der Sensormatrix

Um das frei definierbare Messfeld (Firmware-Versionen < 43, siehe Kapitel 6.13), den Auswahlbereich Messfeld 1/2, einen inversen Auswahlbereich Messfeld (RONI) oder den Referenzbereich für die automatische Belichtungsregelung zu setzen, muss der Prozentwert in den entsprechenden Matrix-Wert konvertiert werden. Die zu verwendende Formel hängt vom entsprechenden Sensor-Typ ab.

```
// Prozentualer Anteil Messfeld
double start_z = 25.0;
double end_z = 75.0;
double start_x = 20.0;
double end_x = 80.0;

// scanCONTROL 30xx
unsigned short col_start = start_x / 100 * 65535;
unsigned short col_size = (end_x - start_x) / 100 * 65535;
unsigned short row_start = start_z / 100 * 65535;
unsigned short row_size = (end_z - start_z) / 100 * 65535;

// scanCONTROL 29xx, 27xx, 25xx
unsigned short col_start = 65535 - (end_x / 100 * 65535);
unsigned short col_size = (end_x - start_x) / 100 * 65535;
unsigned short row_start = 65535 - (end_z / 100 * 65535);
unsigned short row_size = (end_z - start_z) / 100 * 65535;

// scanCONTROL 26xx
unsigned short col_start = 65535 - (end_x / 100 * 65535);
unsigned short col_size = (end_x - start_x) / 100 * 65535;
unsigned short row_start = start_z / 100 * 65535;
unsigned short row_size = (end_z - start_z) / 100 * 65535;
```

Siehe SDK-Beispiele (SetROIs).

6.13 Frei definierbares Messfeld setzen

Dieses Beispiel zeigt, wie ein frei definierbares Messfeld gesetzt werden kann. Dies ermöglicht eine flexible Definition der Messfeldgröße. Das Setzen geschieht über das sequenzielle EXTRA_PARAMETER-Register. Um die Einstellungen zu aktivieren, muss eine 0 in FEATURE_FUNCTION_EXTRA_PARAMETER-Register geschrieben werden.

```
int toggle = 0;

// Aktiviere freies Messfeld
pLLT->SetFeature(FEATURE_FUNCTION_ROI1_PRESET, MEASFIELD_ACTIVATE_FREE);

WriteCommand(0, 0); // Reset
WriteCommand(0, 0); // Initialisierung
WriteCommand(2, 8); // Navigiere in Register
WriteValue2Register(row_start);
WriteValue2Register(row_size);
WriteValue2Register(col_start);
WriteValue2Register(col_size);
WriteCommand(0, 0); // Stop

// Schreibkommande für seq. Register
static void WriteCommand(unsigned int command, unsigned int data)
{
   pLLT->SetFeature(FEATURE_FUNCTION_EXTRA_PARAMETER, (unsigned int)(command << 9)
                                                      + (toggle << 8) + data);

   if (toggle == 1) : toggle = 0 ? toggle = 1;
}

// Schreibe Wert auf Registerposition
static void WriteValue2Register(unsigned short value)
{
   WriteCommand(1, (unsigned int)(value/256));
   WriteCommand(1, (unsigned int)(value%256));
}
```

Siehe API: SetFeature(), FEATURE_FUNCTION_ROI1_PRESET, FEATURE_FUNCTION_EXTRA_PARAMETER
Siehe Operational Manual Part B: *OpManPartB.html#roi1*, *OpManPartB.html#extraparameter*

Siehe Kapitel 6.12 zur Berechnung der korrekten Werte für den Auswahlbereich.

6.14 Auswahlbereich Messfeld 1 setzen (ROI1)

Dieses Beispiel zeigt, wie man den Auswahlbereich Messfeld 1 (ROI1) setzt. Diese Einstellung ermöglicht es, eine eigene Messfeldgröße zu bestimmen.

```
// ROI1 aktivieren
pLLT->SetFeature(FEATURE_FUNCTION_ROI1_PRESET, 0x82000800);

// Werte auf Sensor schreiben
pLLT->SetFeature(FEATURE_FUNCTION_ROI1_DISTANCE, (row_start << 16) + row_size);
pLLT->SetFeature(FEATURE_FUNCTION_ROI1_POSITION, (col_start << 16) + col_size);

// Aktivieren: 0 auf Extraparameter Register schreiben
pLLT->SetFeature(FEATURE_FUNCTION_EXTRA_PARAMETER, 0);
```

Siehe API: SetFeature(), FEATURE_FUNCTION_ROI1_PRESET, FEATURE_FUNCTION_EXTRA_PARAMETER
Siehe Operation Manual Part B: *OpManPartB.html#roi1*, *OpManPartB.html#extraparameter*

Siehe Kapitel 6.12 zur Berechnung der korrekten Werte für den Auswahlbereich.

Achtung: für die scanCONTROL Serien 27xx, 26xx, 29xx (Firmware Version < v43) muss das frei definierbare Messfeld benutzt werden (siehe Kapitel 6.13). Siehe SDK-Beispiele (SetROIs).

6.15 Einbaulagenkalibrierung auf den Sensor spielen

Dieses Beispiel zeigt, wie man die Einbaulage kalibriert. Dies ist nützlich, wenn man eine konstant schiefe Einbaulage hat, man das Profil aber gerade ausgeben lassen will. Es kann der Winkel und eine Verschiebung in X und Z korrigiert werden. Die Funktionen *SetCustomCalibration()* und *ResetCustomCalibration()* sind nicht in der DLL integriert, sondern sind dem ausführlichen Beispielprogramm zur Einbaulagenkalibrierung zu entnehmen.

```
// Sensor Offset and Skalierung (Abhängig von Sensortyp)
double offset = 95.0;
double scaling = 0.002;

// Rotationszentrum und Winkel
double center_x = -9; // mm
double center_z = 86.7; // mm
double angle = -45; // °

// Verschiebung Rotationszentrum
double shift_x = 0; // mm
double shift_z = 0; // mm

// Setze Kalibrierung
SetCustomCalibration(center_x, center_z, angle, shift_x, shift_z, offset, scaling);

// Reset Kalibrierung
ResetCustomCalibration();
```

Siehe Beispielspielprogramm: Calibration

Siehe SDK-Beispiele (Calibration).

6.16 CMM-Trigger nutzen

Mit dem CMM-Trigger kann der scanCONTROL Sensor der Peripherie (z.B. einer Koordinatenmessmaschine) über ein Triggersignal den genauen Zeitpunkt der Profilerfassung mitteilen. Dies geschieht über die RS422-Schnittstelle des Sensors.

Der CMM-Trigger enthält folgende Parameter zur Anpassung des Triggersignals:

- Polarity: Die Polarität des Triggersignales
- Divisor: Triggerteiler (0 Deaktiviert den CMM-Trigger)
- Mark-Space ratio: Das Abtastverhältnis
- Skew Correction: Korrektur des Versatzes (in 0.5µs-Schritten; muss kleiner sein als der halbe Sensorzyklus (1/Profilfrequenz))

```
unsigned int Incounter = 0, CmmCount = 0;
int  CmmTrigger = 0, CmmActive = 0;

std::vector<unsigned char>Timestamp(16);

// Setze RS422 Interface für 26xx/29xx auf CMM-Trigger
unsigned int Interface = MULTI_RS422_CMM;
pLLT->SetFeature(FEATURE_FUNCTION_DIGITAL_IO, Interface);
```

```
// Setze mark space ratio
pLLT->SetFeature(FEATURE_FUNCTION_CMM_TRIGGER, 0x00000401);
// Setze skew correction
pLLT->SetFeature(FEATURE_FUNCTION_CMM_TRIGGER, 0x00000801);
// Setze output port
pLLT->SetFeature(FEATURE_FUNCTION_CMM_TRIGGER, 0x00000c00);
// Setze trigger divisor
pLLT->SetFeature(FEATURE_FUNCTION_CMM_TRIGGER, 0x00000001);

[…] // Übertragung

// Lese CMM-Zeitstempel
Timestamp2CmmTriggerAndInCounter(Timestamp, &Incounter, &CmmTrigger,
                                                    &CmmActive, &CmmCount);
```

Siehe API: SetFeature(), Timestamp2CmmAndInCounter(), FEATURE_FUNCTION_DIGITAL_IO, FEATURE_FUNCTION_CMM_TRIGGER
Siehe Operational Manual Part B: *OpManPartB.html#cmmtrigger, OpManPartB.html#ioconfig*

Zunächst sollte der CMM-Trigger also vollständig konfiguriert werden, bevor er durch Setzen des Divisors aktiv geschaltet wird. Nach der Aktivierung sollte der CMM-Trigger nicht umkonfiguriert werden. Zudem wird der CMM-Trigger nicht in den User-Modes mitgespeichert, so dass er stets durch die verbundene Software konfiguriert werden muss.

Achtung: wird nur für scanCONTROL Serien 27xx/26xx/29xx/30xx unterstützt.

6.17 Profilfolgen speichern

Dieses Beispiel zeigt, wie man per SDK Profilfolgen speichert und damit zur späteren Auswertung zur Verfügung stellt. Die Profile werden als .avi abgespeichert.

```
// Starten der Profilübertragung
pLLT->TransferProfiles(NORMAL_TRANSFER, true);

// Starte Speichern von Profilen als .avi-Datei
pLLT->SaveProfiles(AviFilename, AVI);

// Warten für den Zeitraum der Profilspeicherung (1 s)
Sleep(1000);

// Stoppe Speicherprozedur
pLLT->SaveProfiles(NULL, AVI);

// Stoppen der Profilübertragung
pLLT->TransferProfiles(NORMAL_TRANSFER, false);
```

Siehe API: TransferProfiles(), SaveProfiles()

6.18 Containermode zur Weiterverarbeitung mit BV-Tools

Dieses Beispiel zeigt, wie man die Übertragung so konfiguriert, dass Standard-Bildverarbeitungstools direkt mit dem übertragenen Format arbeiten können.

```
#include <math.h>

[…] // Setup und ContainerBuffer vector definieren

unsigned int ProfileCount = 500;// Anzahl der Profile in einem Bild/Container
unsigned int LostProfiles = 0;
```

```
// Flags für gerade verwendete Auflösung berechnen
double TempLog = 1.0 / log(2.0);
unsigned int ResBits = (unsigned int)floor((log((double)Resolution)*TempLog)+0.5);
// Setzen des Rearrangement-Parameters zur Extrahierung von Z-Daten
// (ohne Timestamp) mit der eingestellten Auflösung; Setzt auch schon die
// Containerbreite
pLLT->SetFeature(FEATURE_FUNCTION_PROFILE_REARRANGEMENT,
    CONTAINER_DATA_Z | CONTAINER_STRIPE_1 | CONTAINER_DATA_LSBF | ResBits << 12));

// Containergröße setzen
pLLT->SetProfileContainerSize(0, ProfileCount);

// Puffer reservieren (Z-KOO hat 2 byte)
ContainerBuffer.resize(Resolution * 2 * ProfileCount);

// Starten der Profilübertragung
pLLT->TransferProfiles(NORMAL_CONTAINER_MODE, true);

// Pollen eines Profiles und Abspeichern in Puffer
// Anm.: Falls noch kein neuer Container seit dem letzten Aufruf angekommen ist
// gibt die Funktion -104 zurück. Ggfls. in einer Schleife abfragen.
pLLT->GetActualProfile(&ContainerBuffer [0], (unsigned int) ContainerBuffer.size(),
                                                CONTAINER, &LostProfiles));

// Stoppen der Profilübertragung
pLLT->TransferProfiles(NORMAL_CONTAINER_MODE, false);
```

Siehe API: SetFeature(), SetProfileContainerSize(), TransferProfiles(), GetActualProfile(), FEATURE_FUNCTION_PROFILE_REARRANGEMENT
Siehe Operational Manual Part B: *OpManPartB.html#profilerearrangement*

6.19 Übertragung von partiellen Profilen

Dieses Beispiel zeigt das Einrichten einer Übertragung von partiellen Profilen. Das übertragene Profil entspricht hier der Profilkonfiguration *PURE_PROFILE* mit eingeschränkter Punktezahl, d.h. es werden nur X/Z-Werte von einem definierten Punktebereich übertragen.

```
// Struct zum Definieren des partiellen Profils
TPartialProfile PartialProfile;

[…] // Init

// Setzen des partiellen Profils
PartialProfile.nStartPoint = 20; // Offset 20 -> Startpunkt = Punkt 21
PartialProfile.nStartPointData = 4; // Datenoffset 4 Bytes -> Beginn X-Daten
PartialProfile.nPointCount = m_uiResolution / 2; // Halbe Auflösung
PartialProfile.nPointDataWidth = 4; // 4 Bytes -> X und Z (je 2 Bytes)

// Reservieren des Profilpuffers
ProfileBuffer.resize(PartialProfile.nPointCount * PartialProfile.nPointDataWidth);

// Übergeben des partiellen Profils
pLLT->SetPartialProfile(PartialProfile);

[…] // Normale Übertragung mit Callback

// Konvertieren des Pufferinhalts in reale Koordinaten
pLLT->ConvertPartProfile2Values(&ProfileBuffer[0], ProfileBuffer.size(),
     &PartialProfile, scanCONTROLType, 0, 1, NULL, NULL, NULL, &ValueX[0],
                                                        &ValueZ[0], NULL, NULL);
```

Siehe API: SetPartialProfile(), TransferProfiles(), GetActualProfile(), ConvertPartProfile2Values()

Eine Änderung der Profilauflösung mit *setResolution()* muss immer vor dem Setzen der PartialProfile-Konfiguration erfolgen, da der Aufruf von *setResolution()* die PartialProfile-Einstellung zurücksetzt.

6.20 Betrieb von mehreren Sensoren

Dieses Beispiel zeigt, wie in einem Programm mit mehreren Sensoren gearbeitet werden kann.

```
std::vector<unsigned int> Interfaces(5);
static CInterfaceLLT* pLLT = nullptr;
static CInterfaceLLT* pLLT2 = nullptr;
HANDLE hProfileEvent = CreateEvent(NULL, true, false, "ProfileEvent");
bool LoadError;

// Erzeugen eines Handles für jeden Ethernet Scanner
pLLT = new CInterfaceLLT("LLT.dll", &LoadError);
pLLT2 = new CInterfaceLLT("LLT.dll", &LoadError);

// Suche verfügbare Interfaces
pLLT->GetDeviceInterfaces(Interfaces, Interfaces.GetLength(0));

// Setzen der Interfaces
pLLT->SetDeviceInterface(Interfaces[0]);
pLLT2->SetDeviceInterface(Interfaces[1]);

// Verbinden mit beiden Scannern
pLLT->Connect();
pLLT2->Connect();

[…]

// Registrierung Callback Scanner 1
pLLT->RegisterCallback(STD_CALL, (void*)NewProfile, pLLT);

// Registrierung Callback Scanner 2
pLLT2->RegisterCallback(STD_CALL, (void*)NewProfile, pLLT2);

[…] // Start der Übertragungen äquivalent

// Callback-Funktion mit Unterscheidung der Sensordaten
void __stdcall NewProfile(const unsigned char* pucData, unsigned int uiSize,
                                                                    void* pUserData)
{
   if (pUserData == pLLT)
   {
      // Daten Sensor 1
   }

   if (pUserData == pLLT2)
   {
      // Daten Sensor 2
   }
}
```

Siehe API: CreateLLTDevice(), GetDeviceInterfacesFast(), SetDeviceInterface(), Connect(), RegisterCallback()

6.21 Fehlermeldungen bei Verbindungsverlust

Dieses Beispiel zeigt, wie ein Callback zur Fehlerbehandlung bei Verbindungsverlust in einer Windows-Form-Anwendung zu registrieren ist.

```
// Registrieren der Fehlermeldung (Anwendungsabhängig)
pLLT->RegisterErrorMsg(/*UINT */Msg, /*HWND */hWnd, /*WPARAM*/ WParam);

BOOL CMessageWnd::OnWndMsg(unsigned int message, WPARAM wParam,
                                          LPARAM lParam, LRESULT* pResult)
{
  if (message == WM_USER)
  {
    if (lParam == ERROR_CONNECTIONLOST)
      // Fehlerbehandlung
  }

  return CWnd::OnWndMsg(message, wParam, lParam, pResult);
}
```

Siehe API: RegisterErrorMsg()

6.22 Temperatur auslesen

Dieses Beispiel zeigt, wie die aktuelle Innentemperatur des Sensors ausgelesen werden kann. Der Wert beschreibt die Temperatur in 0,1 K-Schritten.

```
unsigned int Temperature = 0;

// Vor dem Auslesevorgang muss 0x86000000 auf das Register geschrieben werden
pLLT->SetFeature(FEATURE_FUNCTION_TEMPERATURE, TEMP_PREPARE_VALUE);

// Auslesen der Temperatur
pLLT->GetFeature(FEATURE_FUNCTION_TEMPERATURE, &Temperature);
```

Siehe API: SetFeature(), FEATURE_FUNCTION_TEMPERATURE
Siehe Operational Manual Part B: *OpManPartB.html#temperature*

6.23 Packet Delay berechnen und setzen

Dieses Beispiel zeigt die Bestimmung des minimale bzw. maximalen Packet Delays für einen Sensor in einen Netzwerk mit mehreren Sensoren an einem Switch. Der Packet Delay ist abhängig von folgenden Faktoren: Der eingestellten Paketgröße, dem gegebenen Netzwerk, die zu übertragende Datenmenge und die Anzahl der Sensoren. Die Datenmenge setzt sich dabei aus dem eingestellten Übertragungsmodus und der Profilfrequenz zusammen. Der minimale Delay berechnet sich aus:

$$PD_{min} = (Anzahl\ der\ Sensoren - 1) * \frac{Paketgröße}{Netzwerkübertragungsrate}$$

Der maximale Delay ergibt sich aus:

$$PD_{max} = \left(1000 * \frac{\frac{1000}{Profilfrequenz}}{\frac{KByte\ pro\ Profil * 1024}{Paketgröße} + 1} - \frac{Paketgröße}{Netzwerkübertragungsrate} \right) * 0{,}8$$

Der zu konfigurierende Wert kann zwischen diesen Schranken liegen. Jedem betroffenen Sensor muss ein Packet Delay gesetzt werden. Die Scanner testen dann Übertragungsslots an und senden bei einem freien Slot die Pakete nun die jeweils verzögerten Pakete ab.

Das Setzen des Wertes geschieht wie folgt (hier wird ein Wert von 50 µs eingestellt):

```
unsigned int PacketDelay = 50;

// Setzen des Packet Delays in us
pLLT->SetFeature(FEATURE_FUNCTION_PACKET_DELAY, PacketDelay);
```

Siehe API: SetFeature(), FEATURE_FUNCTION_PACKET_DELAY

7 API

Im Folgenden wird das vollständige API (*Application Program Interface*) aufgelistet. Jede Funktion ist mit ihren Rückgabe- und Parameterwerten beschrieben.

7.1 Instanz-Funktionen

- **CreateLLTDevice ()**

```
unsigned int
CInterfaceLLT::CreateLLTDevice(iInterfaceType);
```

Festlegen und Rückgabe eines Device Handle („Sensorinstanz“) in der DLL für eine scanCONTROL-Kommunikation, abhängig von der Verbindungsschnittstelle.

Parameter

CInterfaceLLT LLT-Klasse
iInterfaceType Art der Schnittstelle (InterfaceType)

Rückgabewert

Neues Device Handle (0x0 oder 0xFFFFFFFF → Fehler: kein Device Handle erstellt)
Standardfehlerwerte

- **GetInterfaceType ()**

```
int
CInterfaceLLT::GetInterfaceType();
```

Abfrage des verwendeten Interfaces für eine Sensorinstanz.

Parameter

CInterfaceLLT LLT-Klasse

Rückgabewert

Wert InterfaceType (0x0 oder 0xFFFFFFFF → Fehler)
Standardfehlerwerte

- **InterfaceType**

  Verfügbare Interfacetypen:

InterfaceType	Wert	Beschreibung
INTF_TYPE_UNKNOWN	0	Wird von *GetInterfaceType*() im Fehlerfall zurückgegeben, für *CreateLLTDevice()* unzulässig.
INTF_TYPE_SERIAL	1	Verbindung mittels serieller Schnittstelle
INTF_TYPE_FIREWIRE	2	Verbindung mittels Firewire (deprecated)
INTF_TYPE_ETHERNET	3	Verbindung mittels Ethernet

- **DelDevice ()**

```
int
CInterfaceLLT::DelDevice();
```

Löschen der Sensorinstanz vor dem Entladen der DLL. Alle eingestellten Parameter bleiben auf dem Scanner erhalten. Die Treibereinstellungen wie *Packetsize*, dem *Buffer count* und der *Profile config* bleiben nicht erhalten.

Parameter

CInterfaceLLT LLT-Klasse

Rückgabewert

Standardrückgabewerte

7.2 Auswahl-Funktionen

- **GetDeviceInterfaces () / GetDeviceInterfacesFast ()**

```
int
CInterfaceLLT::GetDeviceInterfaces(unsigned int[] pInterfaces, int nSize);

int
CInterfaceLLT::GetDeviceInterfacesFast(unsigned int[] pInterfaces, int nSize);
```

Abrufen der am Rechner verfügbaren scanCONTROL device interfaces. Device interfaces repräsentieren die IP-Adressen von verbundenen Sensoren. *GetDeviceInterfacesFast()* (nur für das Ethernet-Interface) ist bei kleineren Ethernet-Netzwerken wesentlich beschleunigt.

Parameter

CInterfaceLLT LLT-Klasse
pInterfaces Array für verfügbare Interfaces (IP; Node-ID)
nSize Größe des Arrays

Rückgabewert

Anzahl der gefundenen device interfaces
Standardfehlerwerte
Spezifische Rückgabewerte:

ERROR_GETDEVINTERFACES_WIN_NOT_SUPPORTED	-250	Funktion steht nur unter Win 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

- **SetDeviceInterface ()**

```
int
CInterfaceLLT::SetDeviceInterface(unsigned int nInterface, int nAdditional);
```

Zuweisen eines scanCONTROL device interfaces zu einer Sensorinstanz in der DLL. Der Zusatzparameter kann die gewünschte Host-IP-Addresse enthalten, was bei mehreren installierten Netzwerkkarten nützlich ist.

Parameter

CInterfaceLLT LLT-Klasse
nInterfaces Interface des zu verbindenden scanCONTROL
nAdditional IP-Adresse des Hosts (optional)

Rückgabewert

Standardrückgabewerte
Spezifischer Rückgabewert:

ERROR_GETDEVINTERFACES_CONNECTED	-252	scanCONTROL ist verbunden, Disconnect() aufrufen

- **SetDiscoveryBroadcastTarget ()**

```
int
CInterfaceLLT::SetDiscoveryBroadcastTarget(unsigned int nNetworkAddress,
                                           unsigned int nSubnetMask);
```

Setzen der Absender-IP-Adresse für den Ethernet-Broadcast (Discovery-Pakete). Nützlich bei mehreren installierten Netzwerkkarten.

Parameter

CInterfaceLLT LLT-Klasse
nNetworkAddress Absender-IP-Adresse
nSubnetMask Absender-Subnetzmaske

Rückgabewert

Standardrückgabewerte

7.3 Verbindungs-Funktionen

- **Connect ()**

```
int
CInterfaceLLT::Connect();
```

Verbinden mit dem ausgewählten scanCONTROL Sensor. Nur möglich, falls mit *SetDeviceInterface()* ein gültiges device interface zugeordnet wurde.

Parameter

CInterfaceLLT LLT-Klasse

Rückgabewert

Standardrückgabewerte
Spezifische Rückgabewerte:

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 -> via *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

- **Disconnect ()**

```
int
CInterfaceLLT::Disconnect();
```

Trennen der Verbindung zum scanCONTROL Sensor. Alle eingestellten Parameter bleiben auf dem Scanner erhalten. Die Treibereinstellungen wie *Packetsize*, dem *Buffer* count und der *Profile config* bleiben nicht erhalten.

Parameter

CInterfaceLLT LLT-Klasse

Rückgabewert

Standardrückgabewerte

7.4 Identifikations-Funktionen

- **GetDeviceName ()**

```
int
CInterfaceLLT::GetDeviceName(char * DevName, unsigned int DevNameSize,
                             char * VenName, unsigned int VenNameSize);
```

Abfrage des Geräte- und Herstellernamens des scanCONTROL Sensors.

Parameter

CInterfaceLLT	LLT-Klasse
DevName	Array für Namen des Devices
DevNameSize	Größe des DevName-Puffers
VenName	Array für Namen des Herstellers
VenNameSize	Größe des VenName-Puffers

Rückgabewert

Standardrückgabewerte

Spezifische Rückgabewerte:

ERROR_GETDEVICENAME_SIZE_TOO_LOW	-1	Die Größe einer der Puffers ist zu klein
ERROR_GETDEVICENAME_NO_BUFFER	-2	Es wurde kein Puffer übergeben

- **GetLLTVersions ()**

```
int
CInterfaceLLT::GetLLTVersions(unsigned int * uiDSP, unsigned int * uiFPGA1,
                              unsigned int * uiFPGA2);
```

Abfrage der Firmware-Version des scanCONTROL Sensors.

Parameter

CInterfaceLLT	LLT-Klasse
uiDSP	Firmware-Version DSP
uiFPGA1	Firmware-Version FPGA1
uiFPGA2	Firmware-Version FPGA2

Rückgabewert

Standardrückgabewerte

- **GetLLTType ()**

```
int
CInterfaceLLT::GetLLTType(TScannerType *ScannerType);
```

Abfrage des Messbereichs und des Typs des scanCONTROL Sensors.

Parameter

CInterfaceLLT LLT-Klasse
ScannerType Scannertyp und Messbereich

Rückgabewert

Standardrückgabewerte

- **ScannerType**

TScannerType	Wert	scanCONTROL Type	Messbereich
StandardType	-1	-	-
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
scanCONTROL29xx_10	3003	29xx	10 mm
scanCONTROL30xx_25	4000	30xx	25mm
scanCONTROL30xx_50	4001	30xx	50mm
scanCONTROL25xx_25	5000	25xx	25mm
scanCONTROL25xx_50	5002	25xx	50mm
scanCONTROL25xx_100	5001	25xx	100mm

7.5 Eigenschafts-Funktionen

7.5.1 Set-/Get-Funktionen

- **GetFeature ()**

```
int
CInterfaceLLT::GetFeature(unsigned int Function, unsigned int * pValue);
```

Auslesen des aktuellen Parameterwertes / Überprüfen der Verfügbarkeit einer Eigenschaft anhand Tabelle in Kapitel 7.5.2.

Parameter

CInterfaceLLT	LLT-Klasse
Function	Registeradresse der Funktion (FEATURE oder INQUIRY)
pValue	Ausgelesener Wert

Rückgabewert

Standardrückgabewerte

Spezifischer Rückgabewert:

ERROR_SETGETFUNCTIONS_WRONG_FEATURE_ADRESS	-155	Die Adresse der gewählten Eigenschaft ist falsch

- **SetFeature ()**

```
int
CInterfaceLLT::SetFeature(unsigned int Function, unsigned int Value);
```

Setzen des Parameters einer Eigenschaft.

Parameter

CInterfaceLLT	LLT-Klasse
Function	Registeradresse der Funktion (FEATURE)
Value	Zu schreibender Wert

Rückgabewert

Standardrückgabewerte

Spezifischer Rückgabewert:

ERROR_SETGETFUNCTIONS_WRONG_FEATURE_ADRESS	-155	Die Adresse der gewählten Eigenschaft ist falsch

7.5.2 Eigenschaften / Parameter

Im Folgenden werden die Eigenschaftsregister erläutert. INQUIRY-Register dienen zur Überprüfung, ob die jeweilige Funktion vorhanden ist. FEATURE-Register dienen zur Abfrage und Einstellung der Werte. Beide Register definieren das LSB bei Bit 0.

Mithilfe des ausgelesenen Wertes eines **INQUIRY-Registers** kann das Setzen eines Features klassifiziert werden. Dazu liefert das Register den minimal und maximal einstellbaren Wert, ob eine automatische Regelung verfügbar ist und ob die Eigenschaft verfügbar ist:

31	30..26	25	24	23..12	11..0
Eigenschaft verfügbar (1 Bit)	Res. (5 Bit)	Auto (1 Bit)	Res. (1 Bit)	Min Wert (12 Bit)	Max Wert (12 Bit)

Der Wert des **FEATURE-Registers** ist mithilfe des Operation Manual Part B zu interpretieren.

Beispiel: Laser

Feature name	Inquiry address	Status and control address	Default setting
Laser	0xfffff0f00524	0xfffff0f00824	0x82000002

Bit	Function
1..0	Laser Power **0** OFF **1** reduced power **2** full power
11	Pulsed Mode Enable The laser is switched on only in the first half of the measurement interval. Additionally the external trigger output is delayed by half of the measurement interval (or 180 degrees). A synchronised slave sensor would measure during the master's idle time. It is recommended to set up Exposure Time < Idle Time.

Abb. 7: Auszug aus dem Operation Manual Part B

Daraus ergibt sich, dass das für die Laserleistung zu beschreibende Register die Adresse 0xf0f00824 besitzt und mittel den Bits 0 und 1 die Laserleistung geregelt werden kann. Dezimal 0 beschreibt die Einstellung *Laser aus*, $1\,|_{\mathrm{dec}}$ die reduzierte und $2\,|_{\mathrm{dec}}$ die volle Laserleistung. Bit 11 aktiviert den Laserpulsmodus.

- **SERIAL_NUMBER**

FEATURE_FUNCTION_SERIAL_NUMBER FEATURE_FUNCTION_SERIAL (deprecated)	0xf0000410

Auslesen der Seriennummer des verbundenen Sensors. Dieses Register kann nur ausgelesen werden.

- **CALIBRATION_SCALE und CALIBRATION_OFFSET**

FEATURE_FUNCTION_CALIBRATION_SCALE	0xf0a00000
FEATURE_FUNCTION_CALIBRATION_OFFSET	0xf0a00004

Auslesen der Skalierung und des Offsets des verbundenen Sensors. Dieses Register kann nur ausgelesen werden.
Achtung: Nur von der scanCONTROL 30xx-Serie unterstützt.

- **LASER**

FEATURE_FUNCTION_LASER FEATURE_FUNCTION_LASERPOWER (deprecated)	0xf0f00824
INQUIRY_FUNCTION_LASER INQUIRY_FUNCTION_LASERPOWER (deprecated)	0xf0f00524

Steuern und Auslesen der Laserleistung: 0 (aus), 1 (reduziert), 2 (voll). Je nach Gerätetyp kann auch die Polarität der externen Laser-Schutzabschaltung oder der Laserpulsmodus eingestellt werden. Das direkt nach der Laserumschaltung übertragene Profil kann korrupt sein.
Siehe *OpManPartB.html#laser* oder *#laserpower* für den verwendeten Sensortyp.

- **ROI1_PRESET**

FEATURE_FUNCTION_ROI1_PRESET FEATURE_FUNCTION_MEASURINGFIELD (deprecated)	0xf0f00880
INQUIRY_FUNCTION_ROI_PRESET INQUIRY_FUNCTION_MEASURINGFIELD (deprecated)	0xf0f00580

Setzen oder Auslesen eines vordefinierten Messfeldes oder Aktivierung der erweiterten Messfeldkonfiguration. Das direkt nach der Messfeldänderung übertragene Profil kann korrupt sein.
Siehe *OpManPartB.html#roi1* oder *zoom* für den verwendeten Sensortyp.
Eine Übersicht über die vordefinierten Messfelder und die damit möglichen maximalen Profilfrequenzen liefert QuickReference.html für den verwendeten Sensortyp.

- **ROI1**

FEATURE_FUNCTION_ROI1_POSITION FEATURE_FUNCTION_FREE_MEASURINGFIELD_X (depr.)	0xf0b0200c
FEATURE_FUNCTION_ROI1_DISTANCE FEATURE_FUNCTION_FREE_MEASURINGFIELD_Z (depr.)	0xf0b02008

Setzt Start und Größe in X und Z des ROI 1. Die möglichen Werte reichen von 0 bis 65535. Die Matrixrotation der Sensoren ist dabei zu beachten. Aktiviert wird die Einstellung mittels des Extraparameter-Registers (FEATURE_FUNCTION_EXTRAPARAMETER).
Voraussetzung: Firmware v43 oder neuer (bei Firmware < v43 muss das EXTRA_PARAMETER Register benutzt werden, um das ROI zu setzen).
Siehe *OpManPartB.html#roi1* oder *#extraparameter* für den verwendeten Sensortyp.

- **ROI1_TRACKING**

FEATURE_FUNCTION_ROI1_TRACKING_DIVISOR FEATURE_FUNCTION_DYNAMIC_TRACK_DIVISOR (depr.)	0xf0b02010
FEATURE_FUNCTION_ROI1_TRACKING_FACTOR FEATURE_FUNCTION_DYNAMIC_TRACK_FACTOR (depr.)	0xf0b02014

Setzt die encoderbasierte Messfeldnachverfolgung. Aktiviert wird die Einstellung mittels des Extraparameter-Registers (FEATURE_FUNCTION_EXTRAPARAMETER).
Voraussetzung: Firmware v43 oder neuer (bei Firmware < v43 muss das EXTRA_PARAMETER Register benutzt werden, um das ROI zu setzen).
Siehe *OpManPartB.html#roi1* oder *#extraparameter* für den verwendeten Sensortyp.

- **IMAGE_FEATURES**

FEATURE_FUNCTION_IMAGE_FEATURES	0xf0b02100

Register zur Aktivierung/Deaktivierung von ROI 2, dem Ausschlussbereich (RONI), dem Referenzbereich für die Belichtungsregelung auf der Sensor-Matrix. Setzt den Betriebsmodus des Sensors.
Achtung: Nur von der scanCONTROL 30xx-Serie unterstützt.

Siehe *OpManPartB.html#image_sensor_features* für den verwendeten Sensortyp.

- **ROI2**

FEATURE_FUNCTION_ROI2_POSITION	0xf0b02108
FEATURE_FUNCTION_ROI2_DISTANCE	0xf0b02104

Setzt Start und Größe in X und Z des ROI 1. Die möglichen Werte reichen von 0 bis 65535. Die Matrixrotation der Sensoren ist dabei zu beachten.
Achtung: Nur von der scanCONTROL 30xx-Serie unterstützt.
Siehe *OpManPartB.html#roi2* für den verwendeten Sensortyp.

- **RONI**

FEATURE_FUNCTION_RONI_POSITION	0xf0b02110
FEATURE_FUNCTION_RONI_DISTANCE	0xf0b0210c

Setzt Start und Größe in X und Z des Ausschlussbereichs (RONI). Die möglichen Werte reichen von 0 bis 65535. Die Matrixrotation der Sensoren ist dabei zu beachten.
Achtung: Nur von der scanCONTROL 30xx-Serie unterstützt.
Siehe *OpManPartB.html#roni* für den verwendeten Sensortyp.

- **TRIGGER**

FEATURE_FUNCTION_TRIGGER	0xf0f00830
INQUIRY_FUNCTION_TRIGGER	0xf0f00530

Setzen und Auslesen der Triggereinstellung. Das direkt nach der Änderung der Triggerkonfiguration übertragene Profil kann korrupt sein. Zusammen mit der Triggerfunktion muss meist auch die Trigger-Schnittstelle parametriert werden (siehe *DIGITAL_IO*). Durch Änderung der Triggereinstellung wird auch der Profilzähler zurückgesetzt.
Siehe *OpManPartB.html#trigger* für den verwendeten Sensortyp.

- **EXPOSURE_TIME**

FEATURE_FUNCTION_EXPOSURE_TIME FEATURE_FUNCTION_SHUTTERTIME (deprecated)	0xf0f0081c
INQUIRY_FUNCTION_EXPOSURE_TIME INQUIRY_FUNCTION_SHUTTERTIME (deprecated)	0xf0f0051c

Setzen und Auslesen der Belichtungszeit in 10 µs-Schritten. Der Wert kann zwischen 1 und 4095 liegen. Optional kann hier auch die automatische Belichtungsregelung eingestellt werden.
Siehe *OpManPartB.html#exposuretime* oder *#shutter* für den verwendeten Sensortyp.

- **EA_REFERENCE_REGION**

FEATURE_FUNCTION_EA_REFERENCE_REGION_POSITION	0xf0b02118
FEATURE_FUNCTION_EA_REFERENCE_REGION_DISTANCE	0xf0b02114

Setzt Start und Größe in X und Z des Referenzbereichs für die Autobelichtung. Die möglichen Werte reichen von 0 bis 65535. Die Matrixrotation der Sensoren ist dabei zu beachten.
Achtung: Nur von der scanCONTROL 30xx-Serie unterstützt.
Siehe *OpManPartB.html#exposureautomatic* für den verwendeten Sensortyp.

- **EXPOSURE_AUTOMATIC_LIMITS**

FEATURE_FUNCTION_EXPOSURE_AUTOMATIC_LIMITS	0xf0f00834
INQUIRY_FUNCTION_EXPOSURE_AUTOMATIC_LIMITS	0xf0f00534

Setzen und Auslesen der Limits für die Belichtungsautomatik in 10 µs-Schritten. Der Wert kann zwischen 1 und 4095 liegen.
Achtung: Nur von der scanCONTROL 30xx-Serie unterstützt.
Siehe *OpManPartB.html#exposureautomatic* für den verwendeten Sensortyp.

- **IDLE_TIME**

FEATURE_FUNCTION_IDLE_TIME FEATURE_FUNCTION_IDLETIME (deprecated)	0xf0f00800
INQUIRY_FUNCTION_IDLE_TIME INQUIRY_FUNCTION_IDLETIME (deprecated)	0xf0f00500

Setzen und Auslesen der Totzeit zwischen den Belichtungsintervallen in 10 µs-Schritten. Der Wert kann zwischen 1 und 4095 liegen. Ist die automatische Belichtungsregelung aktiv, wird die Totzeit automatisch so angepasst, dass die Profilfrequenz stabil bleibt
Siehe *OpManPartB.html#idletime* für den verwendeten Sensortyp.

- **PROFILE_PROCESSING**

FEATURE_FUNCTION_PROFILE_PROCESSING FEATURE_FUNCTION_PROCESSING_PROFILEDATA (depr.)	0xf0f00804
INQUIRY_FUNCTION_PROFILE_PROCESSING INQUIRY_FUNCTION_PROCESSING_PROFILEDATA (depr.)	0xf0f00504

Abfragen und Setzen der Einstellung für die Profilverarbeitung, wie z.B. Deaktivieren der Kalibrierung, Spiegelung des Profils, Messdatenverarbeitung (Post-Processing), Reflexionsauswahl oder erweiterte Belichtungseinstellung.
Siehe *OpManPartB.html#profileprocessing* oder *#processingprofile* für den verwendeten Sensortyp.

- **THRESHOLD**

FEATURE_FUNCTION_THRESHOLD	0xf0f00810
INQUIRY_FUNCTION_THRESHOLD	0xf0f00510

Setzen und Auslesen des Schwellwerts für die Messdatenaufnahme. Bei Targets mit mehreren Reflektionen kann das Erhöhen der Schwelle zu besseren Ergebnissen führen. Optional kann hier auch die dynamische Threshold-Regelung aktiviert werden.
Siehe *OpManPartB.html#threshold* für den verwendeten Sensortyp.

- **MAINTENANCE**

FEATURE_FUNCTION_MAINTENANCE FEATURE_FUNCTION_MAINTENANCEFUNCTIONS (depr.)	0xf0f0088c
INQUIRY_FUNCTION_MAINTENANCE INQUIRY_FUNCTION_MAINTENANCEFUNCTIONS (depr.)	0xf0f0058c

Abfragen und Setzen interner Einstellungen, wie z.B. dem Encoderzähler.
Siehe *OpManPartB.html#maintenance* für den verwendeten Sensortyp.

- **ANALOGFREQUENCY**

FEATURE_FUNCTION_ANALOGFREQUENCY	0xf0f00828
INQUIRY_FUNCTION_ANALOGFREQUENCY	0xf0f00528

Analogfrequenz für die Analogausgänge der scanCONTROL 28xx-Serie. 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.
Siehe *OpManPartB.html#focus* für den scanCONTROL 28xx.

- **ANALOGOUTPUTMODES**

FEATURE_FUNCTION_ANALOGOUTPUTMODES	0xf0f00820
INQUIRY_FUNCTION_ANALOGOUTPUTMODES	0xf0f00520

Modes für die Analogausgänge der scanCONTROL 28xx-Serie. Einstellen der Analog output modes. Es können z.B. die Spannungsbereiche und die Polarität der analogen Ausgänge umgeschaltet werden. Siehe *OpManPartB.html#gain* für den scanCONTROL 28xx.

- **CMM_TRIGGER**

FEATURE_FUNCTION_CMM_TRIGGER FEATURE_FUNCTION_CMMTRIGGER (deprecated)	0xf0f00888

INQUIRY_FUNCTION_CMM_TRIGGER INQUIRY_FUNCTION_CMMTRIGGER (deprecated)	0xf0f00588

Konfiguration der optionalen CMM-Trigger-Funktionen. Die Konfiguration des CMM-Triggers erfolgt durch mehrere Schreibzugriffe auf dieses Register. Zurückgelesen werden kann nur der zuletzt geschriebene Wert.
Achtung: wird nur für scanCONTROL Serien 27xx/26xx/29xx/30xx unterstützt.
Siehe *OpManPartB.html#cmmtrigger* für den verwendeten Sensortyp.

- **PROFILE_REARRANGEMENT**

FEATURE_FUNCTION_PROFILE_REARRANGEMENT FEATURE_FUNCTION_REARRANGEMENT_PROFILE (depr.)	0xf0f0080c
INQUIRY_FUNCTION_PROFILE_REARRANGEMENT INQUIRY_FUNCTION_REARRANGEMENT_PROFILE (depr.)	0xf0f0050c

Parametrierung der übertragenen Profilinformationen im Container-Mode.
Siehe *OpManPartB.html#profilerearrangement* oder *#rearrangementprofile* für den verwendeten Sensortyp.

- **PROFILE_FILTER**

FEATURE_FUNCTION_PROFILE_FILTER	0xf0f00818
INQUIRY_FUNCTION_PROFILE_FILTER	0xf0f00518

Anwendung von Resampling, Median-Filter und/oder Average-Filter.
Siehe *OpManPartB.html#profilefilter* für den verwendeten Sensortyp.

- **DIGITAL_IO**

FEATURE_FUNCTION_DIGITAL_IO FEATURE_FUNCTION_RS422_INTERFACE_FUNCTION (depr.)	0xf0f008c0
INQUIRY_FUNCTION_DIGITAL_IO INQUIRY_FUNCTION_RS422_INTERFACE_FUNCTION (depr.)	0xf0f005c0

Parameter für die Modi-Einstellung der RS422-Schnittstelle bzw. den digitalen Schnittstellen.
Siehe *OpManPartB.html#ioconfig bzw. OpManPartB.html#capturesize* für den verwendeten Sensortyp.

- **PACKET_DELAY**

FEATURE_FUNCTION_PACKET_DELAY	0x00000d08

Ethernet-Paketverzögerung für den Betrieb mehrerer Sensoren an einem Switch in µs. Der einzustellende Wert kann zwischen 0 und 1000 µs liegen.

- **TEMPERATURE**

FEATURE_FUNCTION_TEMPERATURE	0xf0f0082c
INQUIRY_FUNCTION_TEMPERATURE	0xf0f0052c

Auslesen der Sensortemperatur in 0,1 K-Schritten. Bevor die aktuelle Temperatur ausgelesen werden kann, muss erst 0x86000000 auf das Feature-Register geschrieben werden. (*OpManPartB.html#temperature*)

- **EXTRA_PARAMETER**

FEATURE_FUNCTION_EXTRA_PARAMETER FEATURE_FUNCTION_SHARPNESS (deprecated)	0xf0f00808
INQUIRY_FUNCTION_EXTRA_PARAMETER INQUIRY_FUNCTION_SHARPNESS (deprecated)	0xf0f00508

Einstellungen für Peak-Filter, frei definierbares Messfeld und Einbaulagenkalibrierung. Die Konfiguration erfolgt durch mehrere Schreibzugriffe auf dieses Register. Zurückgelesen werden kann nur der zuletzt geschriebene Wert. Seit DLL Version 3.7 / Sensor-Firmware v43, hat dieses Register hauptsächlich die Funktion einige gesetzte Registerwerte zu aktivieren.
Siehe *OpManPartB.html#extraparameter* für den verwendeten Sensortyp.

- **PEAKFILTER**

FEATURE_FUNCTION_PEAKFILTER_WIDTH	0xf0b02000
FEATURE_FUNCTION_PEAKFILTER_HEIGHT	0xf0b02004

Setzt die minimal und maximal für eine Reflexion zulässige Intensität bzw. Reflexionsweite. Die Werte reichen von 0 bis 1023.
Voraussetzung: Firmware v43 oder neuer (bei Firmware < v43 muss das EXTRA_PARAMETER Register benutzt werden).

- **FEATURE_FUNCTION_CALIBRATION**

FEATURE_FUNCTION_CALIBRATION_0 - 7	0xf0b02020 - 0xf0b0203c

Setzt Parameter der Einbaulagenkalibrierung. Zur Aktivierung muss 0 auf das EXTRA_PARAMETER Register geschrieben werden.
Voraussetzung: Firmware v43 oder neuer (bei Firmware < v43 muss das EXTRA_PARAMETER Register benutzt werden).

7.6 Spezielle Eigenschafts-Funktionen

7.6.1 Software Trigger

- **TriggerProfile ()**

```
int
CInterfaceLLT::TriggerProfile();
```

Ausführen einer Software-Triggerung, um ein Profil zu erhalten.

Parameter
CInterfaceLLT LLT-Klasse

Rückgabewert
Standardrückgabewerte

- **TriggerContainer ()**

```
int
CInterfaceLLT::TriggerContainer();
```

Ausführen einer Software-Triggerung, um einen Container zu erhalten.

Parameter
CInterfaceLLT LLT-Klasse

Rückgabewert
Standardrückgabewerte

Voraussetzung: Firmware Version v46 oder neuer.
Der Sensor muss sich im Frametrigger-Modus befinden (siehe Digital IO), um diese Funktion nutzen zu können. Andernfalls muss der Trigger-Container-Modus eingestellt werden mit nachfolgenden Funktionen.

- **TriggerContainerEnable ()**
- **TriggerContainerDisable ()**

```
int
CInterfaceLLT::TriggerContainerEnable();

int
CInterfaceLLT::TriggerContainerDisable();
```

Aktivieren/Deaktivieren des Trigger-Container-Modus, um TriggerContainer() ohne eingestellten Frametrigger-Modus (siehe Digital IO) nutzen zu können.

Parameter
CInterfaceLLT LLT class

Rückgabewert
Standardrückgabewerte

7.6.2 Profilkonfiguration

- **GetProfileConfig ()**

```
int
CInterfaceLLT::GetProfileConfig(TProfileConfig * pValue);
```

Abfrage der aktuellen Profilkonfiguration.

Parameter

CInterfaceLLT LLT-Klasse
pValue Ausgelesene eingestellte Profilkonfiguration

Rückgabewert

Standardrückgabewerte

- **SetProfileConfig ()**

```
int
CInterfaceLLT::SetProfileConfig(TProfileConfig Value);
```

Setzen der Profilkonfiguration.

Parameter

CInterfaceLLT LLT-Klasse
Value Zu setzende Profilkonfiguration

Rückgabewert

Standardrückgabewerte
Spezifischer *Rückgabewert*:

ERROR_ SETGETFUNCTIONS_WRONG _PROFILE_CONFIG	-152	Die gewünschte Profilkonfiguration steht nicht zur Verfügung

- **ProfileConfig**

Zur Verfügung stehende *ProfileConfig*-Einstellungen.

Konstante für den Rückgabewert	Wert	Beschreibung
PROFILE	1	Profildaten aller vier Streifen
PURE_PROFILE	2	Reduzierte Profildaten eines Streifens (nur Positions- und Abstandswerte)
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

7.6.3 Profilauflösung / Punkte pro Profil

- **GetResolution ()**

```
int
CInterfaceLLT::GetResolution(unsigned int * pValue);
```

Abfrage der aktuellen Profilauflösung bzw. Messpunkte pro Profil.

Parameter

CInterfaceLLT LLT-Klasse
pValue Ausgelesene eingestellte Profilauflösung

Rückgabewert

Standardrückgabewerte

- **SetResolution ()**

```
int
CInterfaceLLT::SetResolution(unsigned int Value);
```

Setzen der Profilauflösung bzw. Messpunkte pro Profil. Die Auflösung kann nur dann geändert werden, wenn keine Profile übertragen werden. Außerdem werden bei *SetResolution()* alle Einstellungen für das *PartialProfile* gelöscht.

Parameter

CInterfaceLLT LLT-Klasse
Value Zu setzende Profilauflösung

Rückgabewert

Standardrückgabewerte
Spezifischer Rückgabewert:

ERROR_ SETGETFUNCTIONS_NOT _SUPPORTED_RESOLUTION	-153	Die gewünschte Auflösung wird nicht unterstützt

- **GetResolutions ()**

```
int
CInterfaceLLT::GetResolutions(unsigned int[] pValue, int nSize);
```

Abfrage der zur Verfügung stehenden Profilauflösungen.

Parameter

CInterfaceLLT LLT-Klasse
pValue Array für die verfügbaren Profilauflösungen
nSize Größe des übergebenen Arrays

Rückgabewert
Anzahl der verfügbaren Auflösungen
Standardrückgabewerte

ERROR_SETGETFUNCTIONS_SIZE_TOO_LOW	-156	Die Größe des übergebenen Feldes ist zu klein

7.6.4 Container-Größe

- **GetProfileContainerSize ()**

```
int
CInterfaceLLT::GetProfileContainerSize(unsigned int * pWidth,
                                       unsigned int * pHeight);
```

Abfrage der aktuellen Container-Größe.
Parameter

CInterfaceLLT	LLT-Klasse
pWidth	Ausgelesene eingestellte Containerbreite
pHeight	Ausgelesene eingestellte Containerhöhe

Rückgabewert
Standardrückgabewerte

- **SetProfileContainerSize ()**

```
int
CInterfaceLLT::SetProfileContainerSize(unsigned int Width, unsigned int Height);
```

Setzen der Container-Größe. Die Breite wird automatisch beim Aufruf von *SetFeature(FEATURE_FUNCTION_PROFILE_REARRANGEMENT)* gesetzt. Die Höhe kann frei zwischen 0 und der maximal möglichen Höhe gewählt werden und entspricht der Anzahl von Profilen, die in dem Container übertragen werden. Die Container-Höhe sollte nicht höher als die dreifache Profilrate sein.
Ist „Verbinden von aufeinanderfolgenden Profilen“ aktiviert (siehe *OpManPartB.html#rearrangementprofile* oder *OpManPartB.html#profilerearrangement)*, 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.

Parameter

CInterfaceLLT	LLT-Klasse
Width	Zu setzende Containerbreite
Height	Zu setzende Containerhöhe

Rückgabewert
Standardrückgabewerte
Spezifische Rückgabewerte:

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

- **GetMaxProfileContainerSize ()**

```
int
CInterfaceLLT::GetMaxProfileContainerSize(unsigned int * pMaxWidth,
                                          unsigned int * pMaxHeight);
```

Abfrage der maximal möglichen Container-Größe. Ist die maximale Breite 64, so wird der Container-Mode nicht von dem scanCONTROL unterstützt.

Parameter

CInterfaceLLT LLT-Klasse
pMaxWidth Maximale einstellbare Containerbreite
pMaxHeight Maximale einstellbare Containerhöhe

Rückgabewert

Standardrückgabewerte

7.6.5 Haupt-Reflexion

- **GetMainReflection ()**

```
int
CInterfaceLLT::GetMainReflection(unsigned int * pValue);
```

Abfrage der Hauptreflexion, die bei den Profilkonfigurationen *PURE_PROFILE* oder *QUARTER_PROFILE* extrahiert wird.

Parameter

CInterfaceLLT LLT-Klasse
pValue Ausgelesener Wert der Hauptreflexion

Rückgabewert

Standardrückgabewerte

- **SetMainReflection ()**

```
int
CInterfaceLLT::SetMainReflection(unsigned int Value);
```

Setzen der Hauptreflexion („Streifen“), aus dem die Profildaten bei den Profilkonfigurationen *PURE_PROFILE* oder *QUARTER_PROFILE* extrahiert werden. Der Index des auszugebenden Streifens geht von 0 für den 1. Streifen bis 3 für den 4. Streifen.

Parameter

CInterfaceLLT LLT-Klasse
Value Zu setzender Wert der Hauptreflexion

Rückgabewert
Standardrückgabewerte
Spezifischer Rückgabewert:

ERROR_SETGETFUNCTIONS_REFLECTION_NUMBER_TOO_HIGH	-154	Der Index des auszugebenden Streifens ist größer 3

7.6.6 Anzahl der Puffer

Eine hohe Pufferanzahl ist bei sehr hohen Profilfrequenzen, langsamen Rechnern und/oder Rechnern bei denen mehrere Programme im Hintergrund laufen sinnvoll. Bei Container-Mode- oder Videobildübertragungen sind max. 4 Puffer sinnvoll.

- **GetBufferCount ()**

```
int
CInterfaceLLT::GetBufferCount(unsigned int * pValue);
```

Abfrage der Anzahl der Puffer im Treiber für die Datenübertragung.

Parameter
CInterfaceLLT LLT-Klasse
Value Ausgelesene Pufferanzahl

Rückgabewert
Standardrückgabewerte

- **SetBufferCount ()**

```
int
CInterfaceLLT::SetBufferCount(unsigned int Value);
```

Setzen der Anzahl der Puffer im Treiber für die Datenübertragung.
Parameter
CInterfaceLLT LLT-Klasse
Value Zu setzende Pufferanzahl

Rückgabewert
Standardrückgabewerte
Spezifischer Rückgabewert:

ERROR_SETGETFUNCTIONS_WRONG_BUFFER_COUNT	-150	Die Anzahl der gewünschten Puffer liegt nicht im Bereich >=2 und <= 200

7.6.7 Vorgehaltene Puffer für das Profile-Polling

- **GetHoldBuffersForPolling ()**

```
int
CInterfaceLLT::GetHoldBuffersForPolling(unsigned int * HoldBuffersForPolling);
```

Abfrage der Anzahl der vorgehaltenen Puffer für das Abholen mit *GetActualProfile()*.

Parameter

CInterfaceLLT	LLT-Klasse
HoldBuffersForPolling	Zu setzende Pufferanzahl für Polling

Rückgabewert

Standardrückgabewerte

- **SetHoldBuffersForPolling ()**

```
int
CInterfaceLLT::SetHoldBuffersForPolling(unsigned int HoldBuffersForPolling);
```

Setzen der Anzahl der vorgehaltenen Puffer für das Abholen von Profilen/Containern mit *GetActualProfile().* Der Puffer arbeitet nach dem FIFO-Prinzip. Je größer die Anzahl ist, desto mehr Profile werden zwischengespeichert und die Häufigkeit von Profilausfällen beim Abholen mit *GetActualProfile()* wird verringert. Die Anzahl kann maximal halb so groß wie die Anzahl der Puffer im Treiber sein. Defaultwert: 1.

Parameter

CInterfaceLLT	LLT-Klasse
HoldBuffersForPolling	Ausgelesene Pufferanzahl für Polling

Rückgabewert

Standardrückgabewerte

Spezifischer Rückgabewert:

ERROR_SETGETFUNCTIONS_WRONG_BUFFER_COUNT	-150	Die Anzahl der gewünschten Puffer liegt nicht im Bereich >=2 und <= 200

7.6.8 Paketgröße

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.

- **GetPacketSize ()**

```
int
CInterfaceLLT::GetPacketSize(unsigned int * pValue);
```

Abfrage der aktuellen Paketgröße für die Größe der Ethernet-Streaming-Pakete.

Parameter

CInterfaceLLT	LLT-Klasse
pValue	Ausgelesene Paketgröße

Rückgabewert

Standardrückgabewerte

- **SetPacketSize ()**

```
int
CInterfaceLLT::SetPacketSize(unsigned int Value);
```

Setzen der aktuellen Paketgröße für die Größe der Ethernet Streaming Pakete. Diese Paketgröße muss zwischen der minimalen und maximalen Paketgröße liegen.

Parameter

CInterfaceLLT	*LLT-Klasse*
Value	Zu setzende Paketgröße

Rückgabewert

Standardrückgabewerte

Spezifischer Rückgabewert:

ERROR_SETGETFUNCTIONS_PACKET_SIZE	-151	Die gewünschte Paketgröße wird nicht unterstützt

- **GetMinMaxPacketSize ()**

```
int
CInterfaceLLT::GetMinMaxPacketSize(unsigned long * pMinPacketSize,
                                   unsigned long * pMaxPacketSize);
```

Abfragen der minimalen und maximalen Paketgröße der Ethernet Streaming Pakete.

Parameter

CInterfaceLLT	*LLT-Klasse*
pMinPacketSize	Minimal einstellbare Paketgröße
pMaxPacketSize	Maximal einstellbare Paketgröße

Rückgabewert

Standardrückgabewerte

7.6.9 Laden und Speichern von Parametersätzen

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 (und damit nicht beschrieben) werden, da er die Standardeinstellungen enthält.

- **GetActualUserMode ()**

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

Abfrage des zuletzt geladenen User-Modes/Parametersatzes. Die scanCONTROL 25xx-, 27xx-, 26xx-, 29xx- und 30xx- Serie unterstützt 16 Usermodes.

Parameter

CInterfaceLLT	LLT-Klasse
pActualUserMode	Gerade geladener Usermode
pUserModeCount	Insgesamt verfügbare Usermodes

Rückgabewert

Standardrückgabewerte

- **ReadWriteUserModes ()**

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

Laden oder Speichern eines User-Modes/Parametersatzes. Ist *nWrite* 0, wird der mit *nUserMode* angegebene Usermode geladen, ansonsten werden die aktuellen Einstellungen unter diesem Usermode gespeichert. Nach dem Laden eines User Modes wird ein Reconnect mit dem Sensor benötigt.

Parameter

CInterfaceLLT	LLT-Klasse
nWrite	Laden (0) oder Schreiben (sonst) eines Usermodes
nUserMode	Zu ladender bzw. schreibender Usermode

Rückgabewert

Standardrückgabewerte

Spezifische Rückgabewerte:

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)

7.6.10 Timeout für die Kommunikationsüberwachung zum Sensor

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 die Kommunikation automatisch vom Sensor aus abgebrochen. Beim Debuggen einer programmierten Anwendung ist oftmals ein zu klein gesetzter Heartbeat-Timeout die Ursache für Verbindungsabbrüche.

- **GetEthernetHeartbeatTimeout ()**

```
int
CInterfaceLLT::GetEthernetHeartbeatTimeout(unsigned int * pValue);
```

Abfrage des eingestellten Verbindungs-Timeouts.

Parameter

CInterfaceLLT LLT-Klasse
pValue Ausgelesener Heartbeat Timeout

Rückgabewert
Standardrückgabewerte

- **SetEthernetHeartbeatTimeout ()**

```
int
CInterfaceLLT::SetEthernetHeartbeatTimeout(unsigned int Value);
```

Setzen des Verbindungs-Timeouts in ms. Der Heartbeat-Timeout kann zwischen 500 und 1.000.000.000 ms liegen.

Parameter
CInterfaceLLT LLT-Klasse
Value Zu setzender Heartbeat Timeout

Rückgabewert
Standardrückgabewerte
Spezifischer Rückgabewert:

ERROR_SETGETFUNCTIONS_HEARTBEAT_TOO_HIGH	-162	Der Parameter für den Heartbeat Timeout ist zu groß

7.6.11 Setzen der Dateigröße für das Speichern von Profilen

- **GetMaxFileSize ()**

```
int
CInterfaceLLT::GetMaxFileSize(unsigned int * pValue);
```

Abfrage der eingestellten maximalen Dateigröße beim Speichern von Profilen in Byte.

Parameter
CInterfaceLLT LLT-Klasse
pValue Ausgelesene maximale Dateigröße

Rückgabewert
Standardrückgabewerte

- **SetMaxFileSize ()**

```
int
CInterfaceLLT::SetMaxFileSize(unsigned int Value);
```

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

Parameter
CInterfaceLLT LLT-Klasse

Value Zu setzende maximale Dateigröße

Rückgabewert
Standardrückgabewerte

7.7 Registrierungs-Funktionen

7.7.1 Registrieren des Callbacks für Profilübertragung

Nach der Registrierung eines Callbacks wird dieser beim Empfang eines Profils/Containers aufgerufen. Sie besitzen als Parameter einen Pointer auf die Profil- /Container-Daten, die dazugehörige Größe des Datenfeldes und einen pUserData-Parameter.
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.

- **RegisterCallback ()**

```
int
CInterfaceLLT::RegisterCallback(TCallbackType tCallbackType,
                        void * tReceiveProfiles, unsigned int pUserData);
```

Registrieren des Callback, der bei Profilankunft aufgerufen wird.

Parameter

CInterfaceLLT	LLT-Klasse
tCallbackType	Aufrufkonvention Callback (0: stdcall; 1: c_decl)
tReceiveProfiles	Zu registrierende Callbackfunktion
pUserData	Nutzerdaten zur Unterscheidung von Sensoren

Rückgabewert
Standardrückgabewerte

- **CallbackType**

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

7.7.2 Registrieren einer Fehlermeldung, die bei Fehlern gesendet wird

- **RegisterErrorMsg ()**

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

Registrieren einer Fehlermeldung.

Parameter

CInterfaceLLT	LLT-Klasse
Msg	Nachricht ID
hWnd	Handle (z.B. Window)
WParam	ID Parameter

Rückgabewert

Standardrückgabewerte

- **WParam (gesendeter Fehler)**

Zur Verfügung stehende Fehlermeldungen:

Konstante für den Rückgabewert	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). *Disconnect()* senden, 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)

7.8 Profilübertragungs-Funktionen

7.8.1 Profilübertragung starten/stoppen

- **TransferProfiles ()**

```
int
CInterfaceLLT::TransferProfiles(TTransferProfileType TransferProfileType,
                                int nEnable);
```

Starten oder stoppen der Profilübertragung. 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.

Parameter

CInterfaceLLT	LLT-Klasse
TransferProfileType	Profilübertragungstyp

nEnable Starten (1) oder Stoppen (0) der Übertragung

Rückgabewert

Größe der empfangenen Profile/ Container
Standardrückgabewerte

- **TransferProfileType**

Zur Verfügung stehende TransferProfileTypes:

Konstante für den Rückgabewert	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)

7.8.2 Übertragung der Matrixansicht / Video Mode starten/stoppen

- **TransferVideoStream ()**

```
int
CInterfaceLLT::TransferVideoStream(TTransferVideoType videoType, int nEnable,
                                   unsigned int * pWidth, unsigned int * pHeight);
```

Starten oder stoppen der Übertragung von Video-Bildern des Bildsensors (Video Mode). Maximal können 25 Bilder pro Sekunde übertragen werden. Video-Bilder können nur mit *GetActualProfile()* abgeholt werden; per Callback stehen sie nicht zur Verfügung.

Parameter

CInterfaceLLT LLT-Klasse
videoType Videoübertragungstyp
nEnable Starten (1) oder Stoppen (0) der Übertragung
pWidth Empfangene Bildweite
pHeight Empfangene Bildhöhe

Rückgabewert

Standardrückgabewerte
Spezifische Rückgabewerte:

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 -> evtl. 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

- **TransferVideoType**

  Zur Verfügung stehende TransferVideoTypes.

Konstante für den Rückgabewert	Wert	Beschreibung
VIDEO_MODE_0	0	Verkleinertes Bild der Matrix
VIDEO_MODE_1	1	Vollständiges Bild der Matrix

7.8.3 Übertragung einer definierten Anzahl von Profilen / Containern

- **MultiShot ()**

```
int
CInterfaceLLT::MultiShot(unsigned int nCount);
```

Anfordern einer definierten Anzahl von Profilen/Containern. Die Anzahl der Profile/Container wird in dem Parameter nCount übergeben. Es können zwischen 1 und 65535 Profile/Container angefordert werden.

Parameter

CInterfaceLLT LLT-Klasse
nCount Anzahl der angeforderten Profile/Container

Rückgabewert

Standardrückgabewerte
Spezifische Rückgabewerte:

ERROR_PROFTRANS_SHOTS_NOT_ACTIVE	-100	Der SHOT_TRANSFER-Mode oder der SHOT_CONTAINER_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

7.8.4 Übertragen eines Profils über die serielle Schnittstelle

- **GetProfile ()**

```
int
CInterfaceLLT::GetProfile();
```

Anfordern eines Profils über die serielle Schnittstelle.

Parameter

CInterfaceLLT LLT-Klasse

Rückgabewert

Standardrückgabewerte

7.8.5 Abholen des aktuellen Profils/Containers/Video-Bildes

- **GetActualProfile ()**

```
int
CInterfaceLLT::GetActualProfile(unsigned char[] pBuffer, int nBuffersize,
                TProfileConfig ProfileConfig, unsigned int * pLostProfiles);
```

Abholen des aktuellen Profils/Containers/Video-Bildes von der LLT.dll.

Parameter

CInterfaceLLT	LLT-Klasse
pBuffer	Übertragungspuffer
nBuffersize	Übertragungspuffergröße
ProfileConfig	Profilkonfiguration der Übertragung
pLostProfiles	Verlorene Profile

Rückgabewert

Anzahl der in den Puffer kopierten Bytes
Standardfehlerwerte
Spezifische Rückgabewerte:

ERROR_PROFTRANS_WRONG_PROFILE_CONFIG	-102	Das geladene Profil kann nicht in die gewünschte Profilkonfiguration konvertieren werden
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

7.8.6 Konvertieren von Profil-Daten

- **ConvertProfile2Values ()**

```
int
CInterfaceLLT::ConvertProfile2Values(const unsigned char[] pProfile,
unsigned int nResolution, TProfileConfig ProfileConfig, ScannerType ScannerType,
      unsigned int nReflection, int nConvertToMM, unsigned short[] pWidth,
           unsigned short[] pMaximum, unsigned short[] pThreshold, double[] pX,
                 double[] pZ, unsigned int[] pM0, unsigned int[] pM1);
```

Extrahieren und Konvertieren von Profil-Daten in Koordinaten und erweiterte Punktinformationen. Die übergebenen Arrays müssen mindestens die Größe der Auflösung (Punkte pro Profil) besitzen.

Parameter

CInterfaceLLT	LLT-Klasse

pProfile	Profilpuffer
nResolution	Punkte pro Profil
ProfileConfig	Profilkonfiguration der Übertragung
ScannerType	Scannertyp
nReflection	Auszuwertender Profilstreifen
nConvertToMM	Konvertieren von X/Z-Werten in Millimeter
pWidth	Array für ausgelesene Punktweiten
pMaximum	Array für ausgelesene Maximalintensitäten
pThreshold	Array für ausgelesene Thresholds
pX	Array für ausgelesene Positionswerte
pZ	Array für ausgelesene Abstandswerte
pM0	Array für ausgelesenes Moment 0
pM1	Array für ausgelesenes Moment 1

Rückgabewert

Standardrückgabewerte
Zusätzliche Rückgabewerte bei Erfolg
Spezifischer Rückgabewert:

ERROR_PROFTRANS_REFLECTION_NUMBER_TOO_HIGH	-110	Die Nummer der gewünschten Streifens ist größer 3

- **ConvertPartProfile2Values ()**

```
int
CInterfaceLLT::ConvertPartProfile2Values(const unsigned char * pProfile,
    TPartialProfile * ProfileConfig, ScannerType ScannerType,
       unsigned int nReflection, int nConvertToMM, ushort[] pWidth,
         ushort[] pMaximum, ushort[] pThreshold, double[] pX, double[] pZ,
                                                    uint[] pM0, uint[] pM1);
```

Extrahieren und Konvertieren von partiellen Profil-Daten in Koordinaten und erweiterte Punktinformationen. Die übergebenen Arrays müssen mindestens die Größe des PointCounts bei *PARTIAL_PROFILE* besitzen.

Parameter

CInterfaceLLT	LLT-Klasse
pProfile	Profilpuffer
PartialProfile	Partielles Profil
ProfileConfig	Profilkonfiguration der Übertragung
ScannerType	Scannertyp
nReflection	Auszuwertender Profilstreifen
nConvertToMM	Konvertieren von X/Z-Werten in Millimeter
pWidth	Array für ausgelesene Punktweiten
pMaximum	Array für ausgelesene Maximalintensitäten
pThreshold	Array für ausgelesene Thresholds
pX	Array für ausgelesene Positionswerte
pZ	Array für ausgelesene Abstandswerte
pM0	Array für ausgelesenes Moment 0
pM1	Array für ausgelesenes Moment 1

Rückgabewert

Standardrückgabewerte

Zusätzliche Rückgabewerte bei Erfolg
Spezifischer Rückgabewert:

ERROR_PROFTRANS_REFLECTION_NUMBER_TOO_HIGH	-110	Die Nummer der gewünschten Streifens ist größer 3

- **Rückgabewerte bei Erfolg**

War der Rückgabewert >0, beschreiben die einzelnen Bits wie die Arrays gefüllt wurden:

Gesetztes Bit	Konstante	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

7.9 Abfrage-Funktionen

- **IsInterfaceType ()**

```
int
CInterfaceLLT::IsInterfaceType(int iInterfaceType);
```

Abfrage des verwendeten Interfaces.

Parameter

CInterfaceLLT LLT-Klasse
iInterfaceType Integerwert des InterfaceTypes

Rückgabewert

Spezifische Rückgabewerte:

IS_FUNC_YES	1	Abgefragter Zustand oder Verbindung ist aktiv
IS_FUNC_NO	0	Abgefragter Zustand oder Verbindung ist nicht aktiv

- **IsTransferingProfiles()**

```
int
CInterfaceLLT::IsTransferingProfiles();
```

Abfrage, ob die Profilübertragung gestartet ist

Parameter

CInterfaceLLT LLT-Klasse

Rückgabewert

Spezifische Rückgabewerte:

IS_FUNC_YES	1	Abgefragter Zustand oder Verbindung ist aktiv
IS_FUNC_NO	0	Abgefragter Zustand oder Verbindung ist nicht aktiv

7.10 Funktionen zur Übertragung von partiellen Profilen

Das Messsystem bietet die Möglichkeit das zu übertragende Profil flexibel einzuschränken. Der Vorteil von diesem Verfahren ist eine geringere Größe der tatsächlich übertragenen Daten. Außerdem können damit nicht benötigte Bereiche eines Profils schon direkt im scanCONTROL verworfen werden.

- **GetPartialProfile ()**

```
int
CInterfaceLLT::GetPartialProfile(TPartialProfile * pPartialProfile);
```

Abfrage der Parameter für die Übertragung von partiellen Profilen.

Parameter

CInterfaceLLT LLT-Klasse
pPartialProfile Referenz auf partielle Profilstruktur

Rückgabewert

Standardrückgabewerte

- **SetPartialProfile ()**

```
int
CInterfaceLLT::SetPartialProfile(TPartialProfile pPartialProfile);
```

Setzen der Parameter für die Übertragung von partiellen Profilen. Alle Parameter der *SetPartialProfile()* Funktion müssen immer ein ganzzahliges Vielfaches der jeweiligen *UnitSize* der Funktion *GetPartialProfileUnitSize()* sein.

Parameter

CInterfaceLLT LLT-Klasse
pPartialProfile Referenz auf zu setzende partielle Profilvariable

Rückgabewert

Standardrückgabewerte
Spezifische Rückgabewerte:

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
ERROR_PARTPROFILE_NOT_MOD_UNITSIZE_DATA	-355	nStartPointData oder PointDataWidth sind kein Vielfaches von nUnitSizePointData

- **GetPartialProfileUnitSize ()**

```
int
CInterfaceLLT::GetPartialProfileUnitSize(unsigned int * pUnitSizePoint,
                                         unsigned int * pUnitSizePointData);
```

Abfrage der verfügbaren Schrittweiten zur Übertragung von partiellen Profilen.

Parameter

CInterfaceLLT	LLT-Klasse
pUnitSizePoint	Ausgelesene UnitSizePoint-Größe
pUnitSizePointData	Ausgelesene UnitSizePointData-Größe

Rückgabewert

Standardrückgabewerte

7.11 Funktionen zur Extrahierung der Timestamp-Informationen

- **Timestamp2TimeAndCount ()**

```
int
Timestamp2TimeAndCount(const unsigned char * pBuffer,
   double * dTimeShutterOpen, double * dTimeShutterClose,
                                       unsigned int * uiProfileCount);
```

Extrahieren der Belichtungsinformationen und des Profilzählers aus dem Timestamp.

Parameter

pBuffer	Referenz auf Timestamp-Bytes des Profilpuffers
dTimeShutterOpen	Ausgelesene Startzeit der Belichtung
dTimeShutterClosed	Ausgelesene Endzeit der Belichtung
uiProfileCount	Ausgelesener Profilzähler

Rückgabewert

Standardrückgabewerte

- **Timestamp2CmmTriggerAndInCounter ()**

```
int
Timestamp2CmmTriggerAndInCounter(const unsigned char * pBuffer,
   unsigned int * pInCounter, int * pCmmTrigger, int * pCmmActive,
                                                  unsigned int * pCmmCount);
```

Extrahieren der optionalen CMM-Trigger-Informationen und des Zählereingangs aus dem Timestamp.

Parameter

pBuffer	Referenz auf Timestamp-Bytes des Profilpuffers
pInCounter	Zählerstand des internen Zählers
pCmmTrigger	*Flag CMM*-Trigger aktiv
pCmmActive	*Flag CMM aktiv*
pCmmCount	Trigger-Zähler

Rückgabewert

Standardrückgabewerte

7.12 Funktionen für das Post-Processing

Das Post-Processing stellt gewisse Module auf dem Sensor zur Verfügung, um Profile auszuwerten. Diese Module stehen nur für scanCONTROL SMART- oder gapCONTROL-Sensoren zur Verfügung.

7.12.1 Post-Processing Parameter lesen und schreiben

- **ReadPostProcessingParameter ()**

```
int
CInterfaceLLT::ReadPostProcessingParameter(unsigned int * pParameter,
                                                    unsigned int nSize);
```

Auslesen der Post-Processing-Parameter.

Parameter

CInterfaceLLT	LLT-Klasse
pParameter	Pointer auf Post-Processing-Parameter-Array
nSize	Größe des Post-Processing-Parameter-Arrays (Max. 1024 DWORDs)

Rückgabewert

Standardrückgabewerte

- **WritePostProcessingParameter ()**

```
int
CInterfaceLLT::WritePostProcessingParameter(unsigned int * pParameter,
                                            unsigned int nSize);
```

Schreiben der Post-Processing-Parameter.

Parameter

CInterfaceLLT	LLT-Klasse
pParameter	Pointer auf Post-Processing-Parameter-Array
nSize	Größe des Post-Processing-Parameter-Arrays (Max. 1024 DWORDs)

Rückgabewert

Standardrückgabewerte

7.12.2 Ergebnisse des Post-Processings extrahieren

- **ConvertProfile2ModuleResult ()**

```
int
CInterfaceLLT::ConvertProfile2ModuleResults(const unsigned char * pProfileBuffer,
    unsigned int nProfileBufferSize, unsigned char * pModuleResultBuffer,
           unsigned int nResultBufferSize, TPartialProfile * pPartialProfile);
```

Extrahieren der Ergebnisse des Post-Processings aus den Profildaten. Sie werden auf die Positions- und Abstands-Koordinaten des vierten Streifens geschrieben.

Parameter

CInterfaceLLT	LLT-Klasse
pProfileBuffer	Referenz auf Profilpuffer
nProfileBufferSize	Größe Profilpuffer
pModuleResultBuffer	Referenz auf Ergebnispuffer
nResultBufferSize	Größe Ergebnispuffer
pPartialProfile	Referenz auf partielles Profil (optional)

Rückgabewert

Anzahl der in den Puffer kopierten Bytes
Standardfehlerwerte
Spezifische Rückgabewerte:

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

7.13 Funktionen zum Laden und Speichern von Profil-Dateien

7.13.1 Speichern von Profilen

- **SaveProfiles ()**

```
int
CInterfaceLLT::SaveProfiles(const char * pFilename, FileType FileType);
```

Speichern von Profilen in eine Datei. Die Profile werden dabei mit der aktuellen Profilkonfiguration gespeichert. Der Dateiname muss inklusive Endung angegeben werden. 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 (vgl. RegisterErrorMsg) gesendet.

Parameter

CInterfaceLLT	LLT-Klasse
pFileName	Name der zu speichernden Datei
FileType	Typ der Datei

Rückgabewert

Standardrückgabewerte
Spezifische Rückgabewerte:

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

- **FileType**

Zur Verfügung stehende *FileTypes*. Es wird empfohlen das AVI-Datenformat zu verwenden, da dieses Format von allen scanCONTROL-Programmen der Micro-Epsilon gelesen werden kann.

FileType	Wert	Beschreibung
AVI	0	AVI-Datei
CSV	1	CSV- Datei (nur für Profile)
BMP	2	BMP- Datei (nur für Video Bilder)
CSV_NEG	3	CSV- Datei (nur für Profile) mit gespiegelter Z-Achse

7.13.2 Laden von Profil-Dateien

- **LoadProfiles ()**

```
int
CInterfaceLLT::LoadProfiles(const char * pFilename,
          TPartialProfile pPartialProfile, TProfileConfig pProfileConfig,
                TScannerType pScannerType, unsigned int * pRearrangementProfile);
```

Laden von Profilen aus einer Datei. Es können *.AVI Dateien geladen werden, welche mit der LLT.dll 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. Zum Beenden des Ladens muss *LoadProfiles(null, null, null, null, null)* aufgerufen werden.
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.

Parameter

CInterfaceLLT	LLT-Klasse
pFileName	Name der zu ladenden Datei
pPartialProfile	Partielles Profil (optional)
pProfileConfig	Profilkonfiguration
pScannerType	Scannertyp
pRearragementProfile	Wert rearrangement register

Rückgabewert

Anzahl von geladenen Profilen/Containern
Standardfehlerwerte
Spezifische Rückgabewerte:

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	Die Referenz auf pRearrengementProfile ist NULL

7.13.3 Navigieren in einer geladenen Profil-Datei

- **LoadProfilesGetPos ()**

```
int
CInterfaceLLT::LoadProfilesGetPos(unsigned int * pActualPosition,
                                  unsigned int * pMaxPosition);
```

Abfrage der Anzahl der Profile in einer Datei und der aktuellen Leseposition.

Parameter

CInterfaceLLT	LLT-Klasse
pActualPosition	Aktuelle Leseposition
pMaxPosition	Maximale Leseposition

Rückgabewert

Standardrückgabewerte

- **LoadProfilesSetPos ()**

```
int
CInterfaceLLT::LoadProfilesSetPos(unsigned int pNewPosition);
```

Setzen der Leseposition in einer geladenen Datei.

Parameter

CInterfaceLLT	LLT-Klasse
pNewPosition	Setzen der Leseposition (0 setzt das erste Profil)

Rückgabewert

Standardrückgabewerte

Spezifischer Rückgabewert:

ERROR_LOADSAVE_FILE_POSITION_TOO_HIGH	-57	Die gewünschte Position ist größer oder gleich der maximalen Position

7.14 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.

Achtung: wird nur für scanCONTROL Serien 27xx/26xx/29xx/30xx unterstützt.

- **StartTransmissionAndCmmTrigger ()**

```
int
CInterfaceLLT::StartTransmissionAndCmmTrigger(unsigned int nCmmTrigger,
   TransferProfileType TransferProfileType, unsigned int nProfilesForerun,
      const unsigned char *  pFilename, TFileType FileType, unsigned int nTimeout);
```

Starten der Profilübertragung mit aktiviertem CMM-Trigger.

Die *StartTransmissionAndCmmTrigger()*-Funktion startet zuerst die Profilübertragung, ohne die Profile dabei per Callback weiterzuleiten. Ist die Verbindung korrekt initialisiert, d.h. die gewünschte Anzahl der Profile ohne einen Ausfall übertragen worden, wird der erste CMMTrigger-Befehl mit dem Divisor an den 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.

Parameter

CInterfaceLLT	LLT-Klasse
nCmmTrigger	Erstes Befehlswort des CMM-Triggers, welches den Divisor und die Polarität enthält
TransferProfileType	Transfereinstellung
nProfilesForerun	Anzahl der kontinuierlich eingegangenen Profile ab der eine stabile Datenübertragung angenommen wird
pFileName	Dateiname für die zu speichernde Datei
pFileType	Dateityp
nTimeout	Timeout in ms für gesamte Funktion

Rückgabewert

Standardrückgabewerte

Spezifische Rückgabewerte:

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 CMM-Trigger angekommen

- **StopTransmissionAndCmmTrigger ()**

```
int
CInterfaceLLT::StopTransmissionAndCmmTrigger(int nCmmTriggerPolarity,
                                             unsigned int nTimeout);
```

Beenden der Profilübertragung mit aktiviertem CMM-Trigger.

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 auch 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.

Parameter

CInterfaceLLT	LLT-Klasse
nCmmTriggerPolarity	Polarität des CMM-Triggers (0 = Low aktiv, 1 = High aktiv)
nTimeout	Timeout in ms für gesamte Funktion

Rückgabewert

Standardrückgabewerte

Spezifischer Rückgabewert:

ERROR_CMMTRIGGER_TIMEOUT_AFTER_SETCMMTRIGGER	- 402	Nach dem setzen des CMMTriggers ist kein Profile mit deaktiviertem CMM-Trigger angekommen

7.15 Fehlerwert Konvertierungs-Funktion

- **TranslateErrorValue ()**

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

Konvertieren eines Fehlerwerts in die zugehörige textuelle Beschreibung.

Parameter

CInterfaceLLT	LLT-Klasse
ErrorValue	Fehlerwert
pString	Puffer für Ausgabe-String
nStringSize	Stringlänge

Rückgabewert

Anzahl der in den Puffer kopierten Zeichen

Standardfehlerwerte

Spezifische Rückgabewerte:

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

7.16 Konfiguration lesen/speichern

- **ExportLLTConfig ()**

```
int
CInterfaceLLT::ExportLLTConfig(const char *fileName);
```

Auslesen aller Parameter und speichern in eine Datei. 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 und kann daher ohne Änderungen mit einem Terminal Programm über die serielle Schnittstelle an das scanCONTROL gesendet werden. Alternativ kann auch ImportLLTConfig verwendet werden.

Parameter

CInterfaceLLT	LLT-Klasse
fileName	Dateiname der Export-Datei

Rückgabewert

Standardrückgabewerte

Spezifischer Rückgabewert:

ERROR_READWRITECONFIG_CANT_CREATE_FILE	-500	Die angegebene Datei kann nicht erstellt werden

- **ExportLLTConfigString ()**

```
int
CInterfaceLLT::ExportLLTConfigString(const char *configData, int size);
```

Auslesen aller Parameter und speichern in einen String. Dieser Konfigurations-String 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 und kann daher ohne Änderungen mit einem Terminal Programm über die serielle Schnittstelle an scanCONTROL gesendet werden. Alternativ kann auch ImportLLTConfigString verwendet werden.

Parameter

CInterfaceLLT	LLT class
configData	Array für Export-String
size	Array Größe

Rückgabewert

Standardrückgabewerte

Rückgabewerte GetFeature()

Spezifischer Rückgabewert:

ERROR_READWRITECONFIG_QUEUE_TO_SMALL	-502	Datenarray zu klein

- **ImportLLTConfig ()**

```
int
CInterfaceLLT::ImportLLTConfig(const char *fileName, bool ignoreCalibration);
```

Lesen und Setzen der von ExportLLTConfig exportierten Parameter. Kann auch .sc1 Dateien einlesen, solange diese mit einer scanCONTROL Configuration Tools Version >=5.2 gespeichert wurde. Das ignore calibration-Flag spezifiziert, ob die Einbaulagenkalibrierung von der Datei mit importiert werden soll.

Parameter

CInterfaceLLT	LLT-Klasse
fileName	Dateiname der Config-Datei
ignoreCalibration	falls wahr, wird Einbaulagenkalibrierung der Datei ignoriert

Rückgabewert

Standardrückgabewerte

Rückgabewerte SetFeature()
Spezifischer Rückgabewert:

ERROR_READWRITECONFIG_CANT_OPEN_FILE	-502	Die angegebene Datei kann nicht geöffnet werden.
ERROR_READWRITECONFIG_FILE_EMPTY	-503	Die angegebene Datei ist leer.
ERROR_READWRITE_UNKNOWN_FILE	-504	Datenformat der Datei falsch.
ERROR_READWRITECONFIG_CANT_CREATE_FILE	-500	Die angegebene Datei kann nicht erstellt werden

- **ImportLLTConfigString ()**

```
int
CInterfaceLLT::ImportLLTConfigString(const char *configData, int size,
                                                    bool ignoreCalibration);
```

Liest Einstellungen die mittels ExportLLTConfigString exportiert wurden und setzt diese auf den Sensor. Das ignore calibration-Flag spezifiziert, ob die Einbaulagenkalibrierung von der Datei mit importiert werden soll.

Parameter

CInterfaceLLT	LLT-Klasse
configData	Array mit Config-String
size	Arraygröße
ignoreCalibration	falls wahr, wird Einbaulagenkalibrierung der Datei ignoriert

Rückgabewert

Standardrückgabewerte
Rückgabewerte SetFeature()
Spezifischer Rückgabewert:

ERROR_READWRITE_UNKNOWN_FILE	-504	Datenformat falsch.

- **SaveGlobalParameter ()**

```
int
CInterfaceLLT::SaveGlobalParameter();
```

Speichern der IP-Einstellungen und der Einbaulagenkalibrierung unabhängig vom User Mode.

Parameter

CInterfaceLLT	LLT-Klasse

Rückgabewert

Standardrückgabewerte

8 Anhang

8.1 Standardrückgabewerte

Alle Funktionen des Interfaces geben einen Integer-Wert als Rückgabewert 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.

Zur Unterscheidung der einzelnen Rückgabewerte stehen mehrere Konstanten zur 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, evtl. 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 scanCONTROLs 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

8.2 Übersicht der Beispiele im SDK

Als Leitfaden für die Integration des scanCONTROLs in eigene Projekte sind die Beispielprogramme im Projektordner gedacht. Sie stehen zur Anschauung komplett mit Quelltext zur Verfügung.

Name	Beschreibung
Calibration	Einbaulagenkalibrierung parametrieren
GetProfiles_Poll	Übertragen von Profilen zur LLT.dll und Einlesen der Profile im Polling Mode via Ethernet
GetProfiles_Callback	Übertragen von Profilen zur LLT.dll und Einlesen der Profile per Callback
MultiShot	Übertragen einer bestimmten Anzahl von Profilen
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
MultiLLT	Verwenden von mehreren scanCONTROL Sensoren in einer Anwendung
CMMTrigger	Verwenden des optionalen programmierbaren Triggers
TriggerProfile	Beispiel mit Software-Profiltriggerung
TriggerContainer	Beispiel mit Software-Containertriggerung
LLTInfo	Gibt Statusinformationen zum angeschlossenen Sensor aus.
SetROIs	ROIs auf der Sensor-Matrix setzen
sC30xx_HighSpeed	Zeigt die Konfiguration des scanCONTROL 30xx im HighSpeed-Modus
ReadPPPResults	Zeigt das Auslesen von Post-Processing-Ergebnissen aus den Profildaten

8.3 Unterstützende Dokumente

[1] Operation Manual PartB 2600: Interface Specification for scanCONTROL 2600 Device Family; Ethernet and Serial Port; Supplement B to the scanCONTROL 2600 Manual; MICRO-EPSILON Optronic GmbH;

[2] Operation Manual PartB 2700: Interface Specification for scanCONTROL 2700 Device

Family; Firewire (IEEE 1394) Bus, Ethernet and Serial Port; Supplement B to the scanCONTROL 2700 Manual; MICRO-EPSILON Optronic GmbH;

[3] Operation Manual PartB 2800: Interface Specification for scanCONTROL 2800 Device Family; Firewire (IEEE 1394) Bus and Serial Port; Supplement B to the scanCONTROL 2800 Manual; MICRO-EPSILON Optronic GmbH;

[4] Operation Manual PartB 2900: Interface Specification for scanCONTROL 2900 Device Family; Ethernet and Serial Port; Supplement B to the scanCONTROL 2900 Manual; MICRO-EPSILON Optronic GmbH;

[5] scanCONTROL 2600 Quick Reference; Brief Introduction to scanCONTROL 2600 Device Family; MICRO-EPSILON Optronic GmbH;

[6] scanCONTROL 2700 Quick Reference; Brief Introduction to scanCONTROL 2700 Device Family; MICRO-EPSILON Optronic GmbH;

[7] scanCONTROL 2800 Quick Reference; Brief Introduction to scanCONTROL 2800 Device Family; MICRO-EPSILON Optronic GmbH;

[8] scanCONTROL 2900 Quick Reference; Brief Introduction to scanCONTROL 2900 Device Family; MICRO-EPSILON Optronic GmbH;

[9] Schnittstellendokumentation zur LLT-DLL; MICRO-EPSILON Optronic GmbH;

[10] Operation Manual PartB 3000: Interface Specification for scanCONTROL 3000 Device Family; Ethernet and Serial Port; Supplement B to the scanCONTROL 3000 Manual; MICRO-EPSILON Optronic GmbH;

[11] scanCONTROL 3000 Quick Reference; Brief Introduction to scanCONTROL 3000 Device Family; MICRO-EPSILON Optronic GmbH;

[12] Operation Manual PartB 2500: Interface Specification for scanCONTROL 2500 Device Family; Ethernet and Serial Port; Supplement B to the scanCONTROL 2500 Manual; MICRO-EPSILON Optronic GmbH;

[13] scanCONTROL 2500 Quick Reference; Brief Introduction to scanCONTROL 2500 Device Family; MICRO-EPSILON Optronic GmbH;