StreamParser-DLL – Schnittstelle zur Anwendungsprogrammierung

StreamParser-DLL V1.1.0

2023-11-23

Doc. Rev. 1.1.0 de-DE

SCANLAB GmbH
Siemensstr. 2a
82178 Puchheim
Deutschland

Tel. +49 (89) 800 746-0
Fax: +49 (89) 800 746-199

info@scanlab.de
www.scanlab.de

Inhalt

1 Über dieses Handbuch

Dieses Handbuch beschreibt die C-API von SCANLAB StreamParser-DLL V1.1.0.

Vorsicht!

- Lesen und befolgen Sie alle Sicherheitshinweise in diesem Handbuch!
- SCANLAB übernimmt keine Haftung für Schäden oder Folgeschäden aufgrund Nichtbeachtung dieses Handbuchs, insbesondere der hierin genannten Sicherheitshinweise.

1.1 Hersteller

SCANLAB GmbH
Siemensstr. 2a
82178 Puchheim
Deutschland
Tel. +49 (89) 800 746-0
Fax: +49 (89) 800 746-199
info@scanlab.de
www.scanlab.de

1.2 Weiterführende Dokumente

- RTC6-Handbuch

1.3 Glossar

API	Abkürzung für Application Programming Interface (“Schnittstelle zur Anwendungsprogrammierung”). Programmteil (hier: der StreamParser-DLL), der anderen Programmen zur Anbindung an das System zur Verfügung gestellt wird (hier: Funktionen der StreamParser-DLL). Siehe Kapitel 5 “In der API bereitgestellte Funktionen”, Seite 26.
Benutzer	Bezeichnet eine Person, die StreamParser-DLL-Anwenderprogramme entwickelt.
BIOS	Basic Input/Output System. Ist im Flash-Speicher der RTC6-Karte dauerhaft gespeichert. Siehe auch `RTC6conf.exe`.
Callback-Event	Eines von mehreren StreamParser-DLL-internen Ereignissen. Siehe auch Callback-Funktion.
Callback-Funktion (Rückruffunktion)	Bezeichnet eine benutzerdefinierte Funktion, die ausgeführt werden soll, sobald (StreamParser-DLL-intern) ein bestimmtes “Callback-Event” auftritt. Siehe **slsp_stream_callback**, Seite 42.
Datenpaket	Nicht: TCP-Paket, siehe “TCP-Paket-Aufbau”, Seite 21. • Entsprechender Proxy (Stellvertreter) ist `struct slsp_rtc_data_package` • Ein Datenpaket abgefragt werden nach: – Anzahl der Aufzeichnungs-Kanäle – Metadaten – Waveform-Daten jedes Aufzeichnungs-Kanals
Datenstrom	• Entsprechender Proxy (Stellvertreter) ist `struct slsp_rtc_data_stream` • Enthält alle gesammelten Datenpakete seit dem letzten Aufruf der Callback-Funktion • Kann nach der Anzahl der Datenpakete abgefragt werden • Jedes Datenpaket kann abgerufen werden
Data-Streaming	Siehe Kurzinformation Data-Streaming, Seite 9.
LSB	Least Significant Bit. Das niedrigstwertige Bit.
Messwertspeicher (RTC-Karte)	Synonym: Waveform-Speicher, Ring-Speicher.
RTC6 Ethernet-Karte	Siehe RTC6-Handbuch.
RTC6-DLL-Anwenderprogramm	Lies: RTC6-DLL-basiertes Anwenderprogramm. Siehe auch Seite 15.

StreamParser	Software-Objekt • Wird erzeugt durch **slsp_stream_parser_create** • Kann über den zurückgegebenen Zeiger `slsp_stream_parser*` angesprochen werden
StreamParser-DLL	Generische Bezeichnung für: • `StreamParser.dll` Win32-basierte dynamische Link Library • `StreamParser_x64.dll` Win64-basierte dynamische Link Library Teil des StreamParser-Softwarepakets.
StreamParser-DLL-Anwenderprogramm	Lies: StreamParser-DLL-basiertes Anwenderprogramm. Siehe auch Seite 15.
StreamParser-Softwarepaket	Siehe Kapitel 2.4 "Softwarepaket-Inhalt", Seite 11.
"Erweiterter Scan-Kopf-Status"	Siehe RTC6-Handbuch.
set_trigger[*]	Lies: "**set_trigger**/**set_trigger4**/**set_trigger8**".
Waveform-Daten	Die in einem Aufzeichnungs-Kanal enthaltenen Daten (wie mit **set_trigger[*]** eingestellt). Siehe auch Nutzdaten, Seite 25.

2 Produktübersicht

In diesem Kapitel:

2.1 Bestimmungsgemäße Verwendung

Die StreamParser-DLL (32-Bit-Version und 64-Bit-Version) ist Teil des StreamParser-Softwarepakets.

Zur Entwicklung von Anwenderprogrammen unter MS Windows stellt sie eine Programmierschnittstelle (API) in Form von Funktionen bereit.

Alleiniger Anwendungskontext dieser Funktionen ist die "Data-Streaming"-Funktionalität, siehe Kurzinformation Data-Streaming, Seite 9, mit den 2 Haupt-Anwendungsfällen:

- "Listen-abhängiger Modus"
  (Daten nur vom Listen-Start bis zu Listen-Ende)
- "Listen-unabhängiger Modus"
  (Daten auch außerhalb der Listen-Abarbeitung, z.B. Temperatur beobachten)

Die StreamParser-DLL-Funktionen ermöglichen[1]:

- Verbindung zur RTC6 Ethernet-Karte aufbauen und trennen
- Entgegennehmen der TCP-Pakete
- Auspacken (= Aufbereiten, Parsen) der TCP-Pakete
- TCP-Paket-Inhalte zur Verfügung stellen (nach Callback-Funktions-Aufruf) für die Weiterverarbeitung

Die StreamParser-DLL-Funktionen können in bestehenden Programmcode verwendet werden und unterstützen somit bei der Entwicklung von kundeneigenen Anwenderprogrammen für:

- Kontinuierliches Aufzeichnen (Loggen) in Dateien oder Datenbanken
- Visualisieren
- Nicht-Echtzeit-Prozess-Steuerung
  (z.B. Laserleistung in verschiedenen Additive Manufacturing-Schichten anpassen)

Für den Zugriff auf alle Funktionalitäten der StreamParser-DLL stehen zur Verfügung, siehe Kapitel 2.4 "Softwarepaket-Inhalt", Seite 11:

- C-API (hier dokumentiert)
- C++-API (objektorientiert)

Vorsicht!

- Mögliche Personenschäden und Sachschäden. Das Zusammenspiel von StreamParser-DLL, RTC6 Ethernet-Karte und Ethernet-Architektur erfüllt *nicht* die Anforderungen an eine Echtzeitfähigkeit (10 µs).
  Wegen der Ethernet-typischen Latenzen im Millisekunden-Bereich ist das rechtzeitige Reagieren (auf eine Datenauswertung im RTC-10 µs-Takt) unmöglich.
  Implementieren Sie deshalb in Ihrem Anwenderprogramm *keine* Sicherheits-Leistungsmerkmale, die auf der StreamParser-DLL basieren!
  Verwenden Sie insbesondere die Ist-Position *nicht* dazu, um den Laser vermeintlich aus Gründen der Sicherheit abzuschalten.

(1) Ohne StreamParser-DLL müssen Benutzer die Ethernet-Pakete selber direkt ("low-level") auspacken und damit arbeiten.

Kurzinformation Data-Streaming

- Hardware-Voraussetzungen
  - i*DRIVE*-Scan-System
    - Keine Mindest-Scan-System-Firmware-Version nötig (im Gegensatz zur Multiplexing-Funktionalität)
    - Gewöhnliches Datenkabel
  - RTC6 Ethernet-Karte
    - ≥ BIOS-ETH 40
    - Keine bestimmte Option erforderlich
    - `RTC6ETH.out` ETH 646
    - `RTC6RBF.rbf` RBF 639
- Software-Voraussetzungen
  - Windows ≥ 10
  - RTC6-Software-Paket ≥ V1.16.3
- Data-Streaming
  - Verbessert die Möglichkeiten zum Übertragen von Daten
  - Benutzer haben nun einen einfacheren Zugang zu diesen Daten
  - Es kann, aber es muss nicht unbedingt eine Liste abgearbeitet werden:
    - "Listen-unabhängiger Modus"
      - Daten auch außerhalb der Listen-Abarbeitung, z.B. Temperatur beobachten
      - Siehe Abbildung 3
    - "Listen-abhängiger Modus"
      - Daten nur vom Listen-Start bis zu Listen-Ende
      - Siehe Abbildung 4
  - Ist eine Funktionalität, bei der die RTC6 Ethernet-Karte (nach einer entsprechenden Konfiguration im "Haupt"-Anwenderprogramm) selbstständig und andauernd Daten (in TCP-Paketen = Ethernet-Datenpaketen verpackt, nicht im 10 μs-Takt, sondern alle n Netzwerkpakete) über eine TCP-Verbindung an einen Empfänger (z.B. StreamParser-DLL-Anwenderprogramm) überträgt, siehe Abbildung 2. Es handelt sich um eine 1:1-Verbindung (nur lesend).

- Inhalt eines TCP-Pakets:
  - "Kopfdaten, Seite 21"
  - "Metadaten, Seite 22"
    - RTC6 Ethernet-Karten-Metadaten z.B. DSP-Versionsnummer (`RTC6ETH.out`)
    - RTC6 Ethernet-Karten-Zustandsinformationen z.B. Unterbrechungs-Status
  - "Nutzdaten, Seite 25"
    = i*DRIVE*-Scan-System-Zustandsdaten
    = Daten, die das i*DRIVE*-Scan-System an die RTC6 Ethernet-Karte überträgt (wird mit **set_trigger[*]** konfiguriert) und dann im RTC6 Ethernet-Karten-Messwertspeicher aufgezeichnet werden, siehe RTC6-Handbuch, **set_trigger[*]** `Signal1`
- StreamParser
  - empfängt die TCP-Pakete
  - parst sie

2.2 Sicherheit

Bei der Entwicklung von StreamParser-DLL-Anwenderprogrammen müssen Sie beachten:

- Sicherheitshinweis Vorsicht!, Seite 5
- Sicherheitshinweis Vorsicht!, Seite 8

2.3 Voraussetzungen

- Hardware-Voraussetzungen, Seite 9
- Software-Voraussetzungen, Seite 9

2.4 Softwarepaket-Inhalt

```
bin         \ StreamParser.dll
              StreamParserd.dll
              StreamParser_x64.dll
              StreamParser_x64d.dll

democode    \ CPP_Demo.cpp
              C_Demo.cpp

doc         \ StreamParser_API_de-DE.pdf
              StreamParser_API_en-US.pdf

include     \ StreamParser.h
              StreamParser_C.h
              StreamParser_Export.h

lib         \ StreamParser.lib
              StreamParserd.lib
              StreamParser_x64.lib
              StreamParser_x64d.lib

Licenses    \ BOOST_LICENSE_1_0.txt
```

3 Softwareentwicklung mit der StreamParser-DLL

In diesem Kapitel:

"Klassisches" Verfahren (= ohne Data-Streaming) – Pull-Mechanismus

Sollen die vom i*DRIVE*-Scan-System an die RTC-Karte übertragenen Signale – wie seine Positionsdaten – aufgezeichnet (geloggt) werden, gilt (ohne Data-Streaming, ohne StreamParser-DLL) folgender Ablauf:

(1) Liste mit **set_trigger[*]**-Aufruf erstellen.

(2) Ausführung der Liste starten.
Sobald **set_trigger[*]** aufgerufen wird:

Die RTC-Karte zeichnet die angefragten Daten in ihrem Messwertspeicher auf.

(3) **get_waveform** aufrufen.
Die aufgezeichneten Daten werden von der RTC-Karte auf den PC übertragen.

(4) Weiterverarbeiten der übertragenen Daten, jedoch zunächst "lokal" = nur in diesem RTC-Anwenderprogramm

Wichtig: Darüber hinaus müssen im ansteuernden RTC-Anwenderprogramm Koordinationsmaßnahmen (wie Messwertspeicher überwachen, Listen laden) programmiert sein.

Verfahren mit Data-Streaming und StreamParser-DLL – Push-Mechanismus

- *Wichtig*:
  - Hardware-Voraussetzungen, Seite 9
  - Software-Voraussetzungen, Seite 9

Die StreamParser-DLL. wird eingebunden, siehe auch Kapitel 3.1 "StreamParser-DLL installieren", Seite 18 in (alternativ):

- Das ansteuernde "Haupt"-Anwenderprogramm, siehe Abbildung 1, Seite 14 (A)
- Ein eigenes Anwenderprogramm (= StreamParser-DLL-Anwenderprogramm), siehe Abbildung 1, Seite 14 (B)
  - Dieses kann wahlweise auf einem anderen PC laufen, siehe Abbildung 1, Seite 14 (C)

Ablauf:

- Siehe Abbildung 2, Seite 15
- Siehe auch Beispiel-Code für RTC6-DLL-Anwenderprogramm
- Siehe auch Beispiel-Code für StreamParser-DLL-Anwenderprogramm

(1) `eth_config_waveform_streaming_ctrl` aufrufen.

Die RTC6 Ethernet-Karte-Data-Streaming-Schnittstelle wird eingeschaltet und ist zum Übertragen bereit (d.h. wartet darauf, dass sich ein TCP-Client mit **slsp_stream_parser_connect**-Aufruf verbindet).

(2) Liste mit **set_trigger[*]**-Aufruf erstellen (= gewünschte Signale angeben).

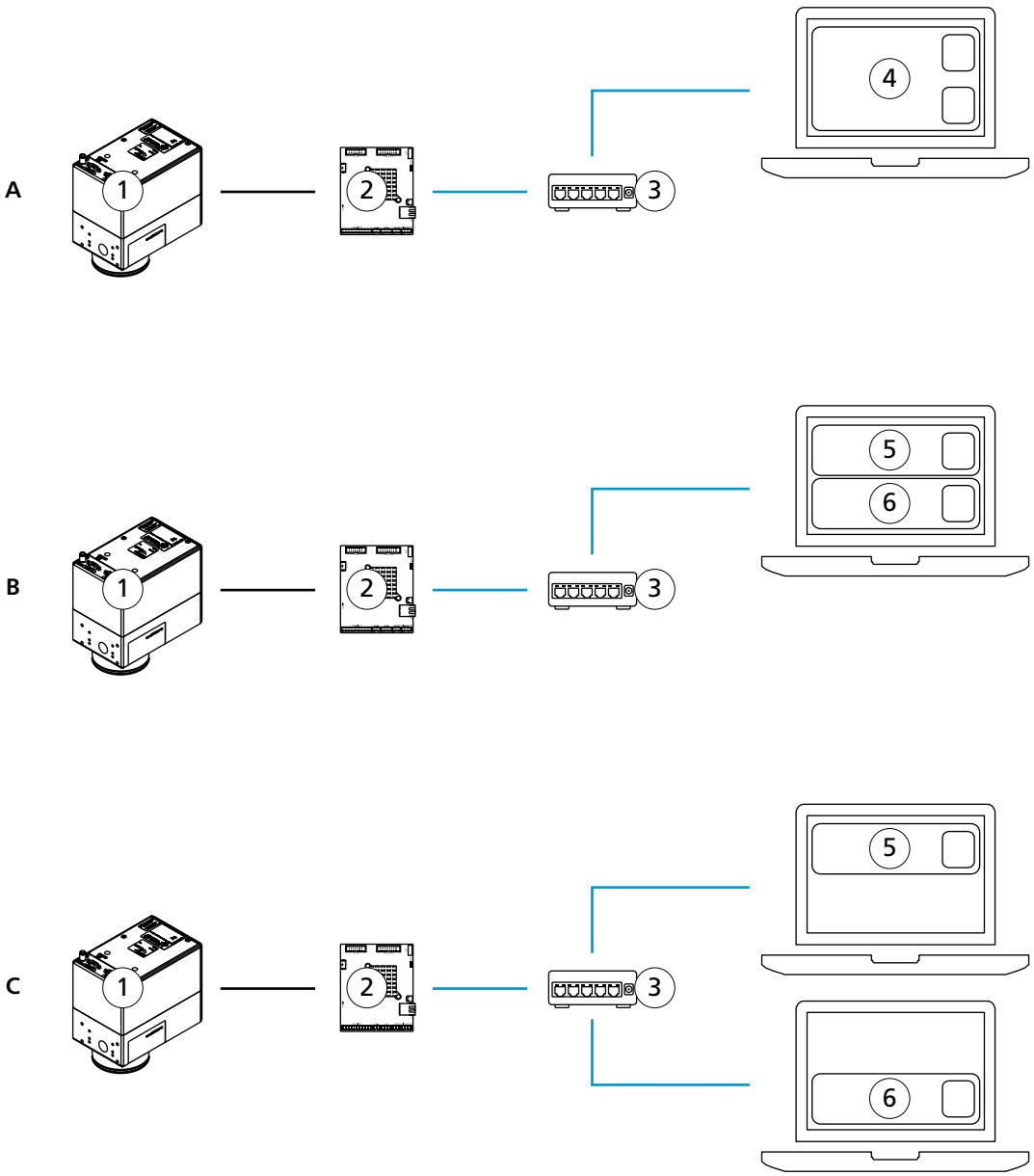
(3) Ausführung der Liste starten.
Sobald **set_trigger[*]** aufgerufen wird:

Die RTC6 Ethernet-Karte überträgt kontinuierlich die Daten (in TCP-Paketen verpackt) über TCP.

(4) Die StreamParser-DLL nimmt die TCP-Pakete entgegen, bereitet sie auf und stellt sie zur Weiterverarbeitung mithilfe einer vom Benutzer implementierten Callback-Funktion zur Verfügung.

Hinweise

- *Wichtig*: Das Weiterverarbeiten (z.B. Aufbereiten von Werten) ist keine Funktionalität der StreamParser-DLL.
- Kein "Listen-unabhängiger Modus" zusammen mit der laserDESK-Protokollfunktionalität.
- Siehe auch Kapitel 3.2 "StreamParser-DLL-Anwenderprogramm entwickeln – Grundsätzliches Vorgehen", Seite 19.
- Hinweise und Beispielcode siehe Kapitel 9 "Beispiel-Code (C++)", Seite 53.



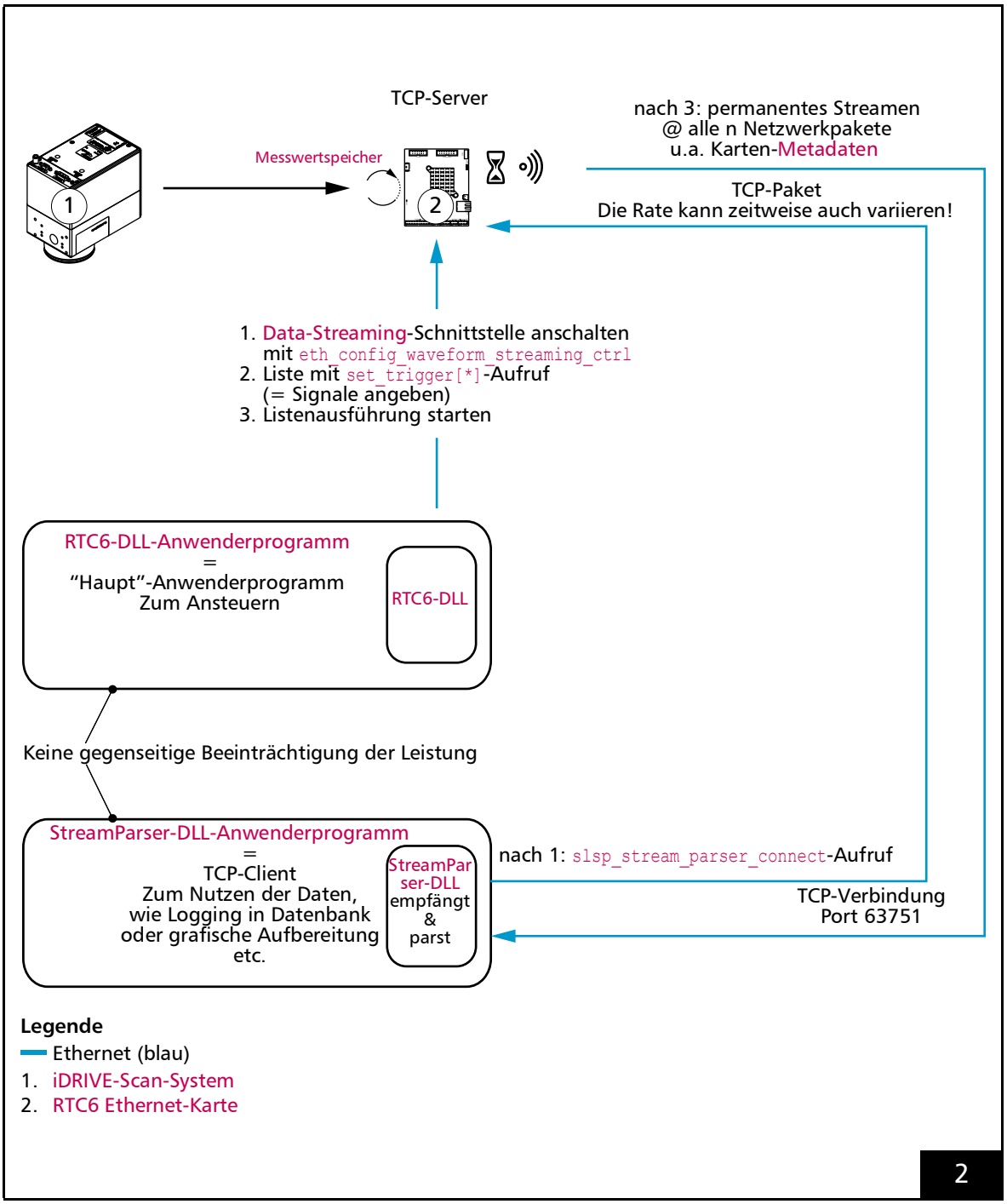
Legende

— Ethernet (blau)

1. iDRIVE-Scan-System
2. RTC6 Ethernet-Karte
3. Switch, non-optional in (C)
4. 1 Anwenderprogramm, basierend auf RTC6-DLL sowie StreamParser-DLL
5. RTC6-DLL-Anwenderprogramm
6. StreamParser-DLL-Anwenderprogramm

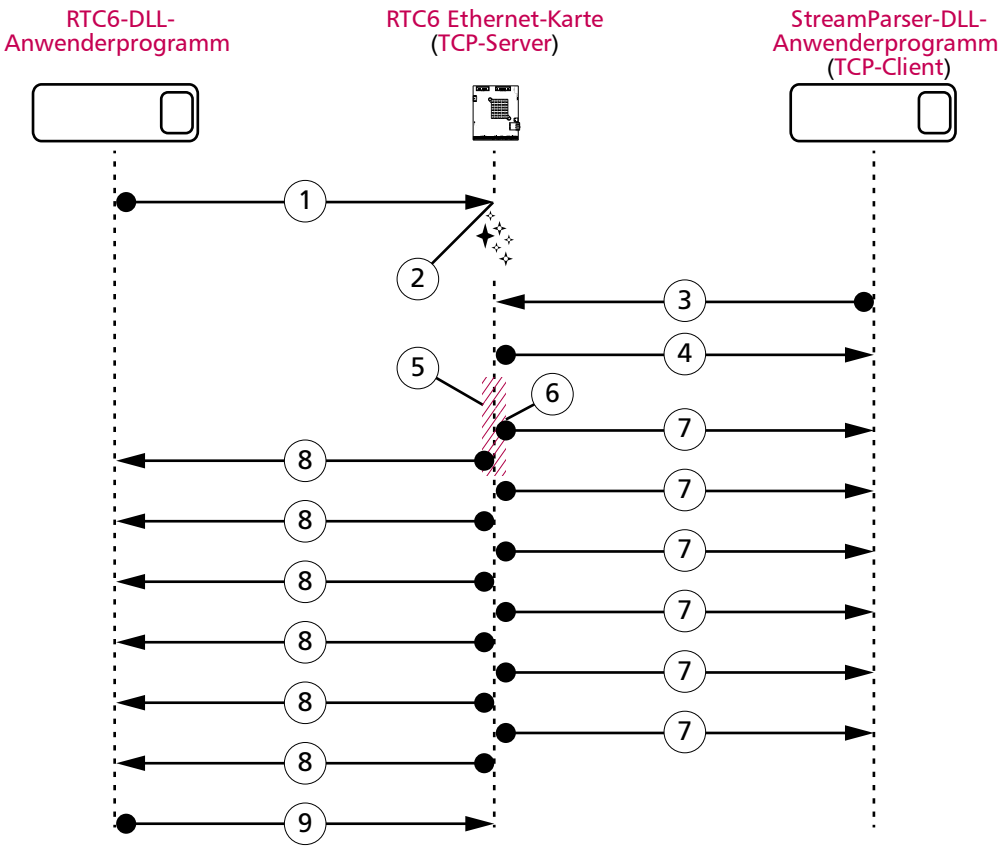
1

Topologien. Siehe Text, Seite 12.



Data-Streaming und StreamParser-DLL. Schema.

"Listen-unabhängiger Modus"



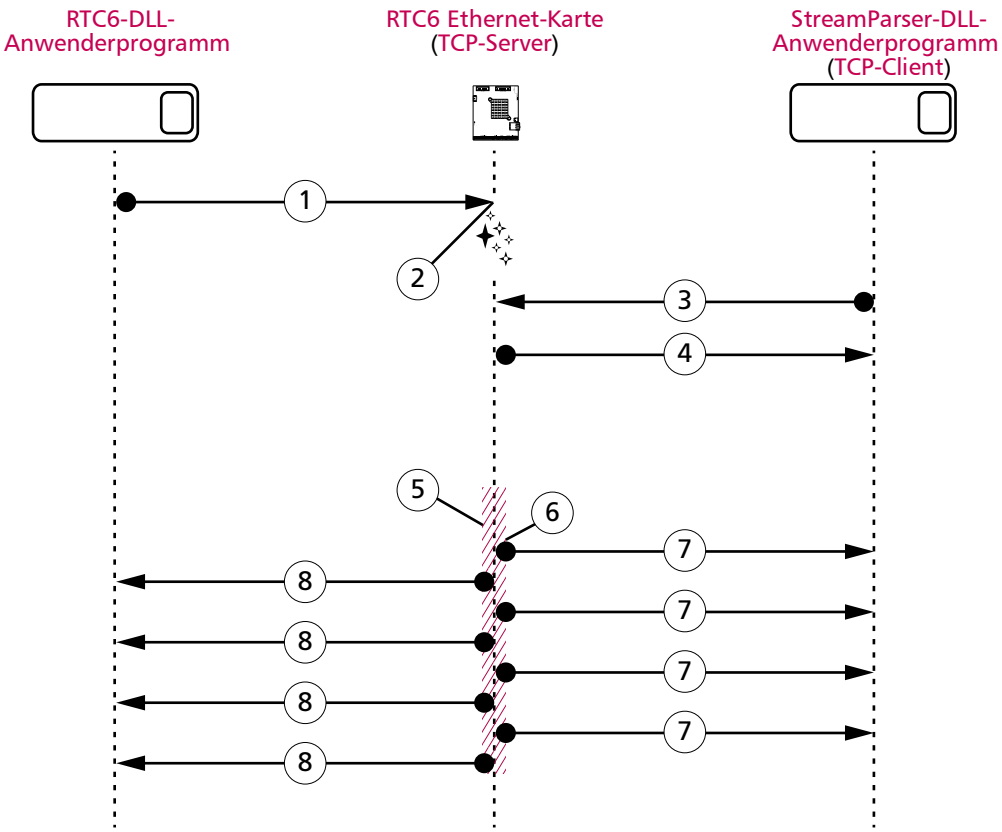
Legende

1. `eth_config_waveform_streaming_ctrl`-Aufruf
2. Bereit zum Annehmen einer TCP-Verbindung
3. TCP Connect
4. TCP Connect OK
5. Liste, wird ausgeführt
6. **set_trigger[*]**-Aufruf. Löst den ersten (7) aus
   - **set_trigger[*]** mit Bit #30 = 1: Kein automatischer Stopp der Aufzeichnung beim Listenende.
   - **set_trigger[*]** mit Bit #31 = 1: Endlos-Aufzeichnung (Ringpuffer).
7. TCP-Paket (gemäß Konfiguration mit **set_trigger[*]**)
8. Datenpaket
9. **stop_trigger**-Aufruf. Zum Beenden

3

StreamParser: "Listen-unabhängiger Modus". Zustands-Diagramm.

"Listen-abhängiger Modus"



Legende

1. `eth_config_waveform_streaming_ctrl`-Aufruf
2. Bereit zum Annehmen einer TCP-Verbindung
3. TCP Connect
4. TCP Connect OK
5. Liste, wird ausgeführt
6. **set_trigger[*]**-Aufruf. Löst den ersten (7) aus
   - **set_trigger[*]** mit Bit #30 = 0: Automatischer Stopp der Aufzeichnung beim Listenende.
   - **set_trigger[*]** mit Bit #31 = 1: Endlos-Aufzeichnung (Ringpuffer).
7. TCP-Paket (gemäß **set_trigger[*]**)
8. Datenpaket

4

StreamParser: "Listen-abhängiger Modus". Zustands-Diagramm.

3.1 StreamParser-DLL installieren

Es gibt verschiedene Möglichkeiten, die StreamParser-DLL in Ihr Anwenderprogramm einzubinden. Im Folgenden ist beispielhaft die Einbindung in ein Projekt beschrieben.

(1) Fügen Sie als Abhängigkeit zu Ihrem Projekt hinzu:

- `StreamParser.h`
- Alternativ (gemäß Anforderungen)
  - Für das 32-Bit-C-Interface
    - `StreamParser.dll` und `StreamParser.lib`
  - Für das 64-Bit-C-Interface
    - `StreamParser_x64.dll` und `StreamParser_x64.lib`
  - Für das 32-Bit-C++-Interface
    - `StreamParser.dll` und `StreamParser.lib` (Release) oder `StreamParserd.dll` und `StreamParserd.lib` (Debug)
  - Für 64-Bit-C++-Interface
    - `StreamParser_x64.dll` und `StreamParser_x64.lib` (Release) oder `StreamParser_x64d.dll` und `StreamParser_x64d.lib` (Debug)

(2) Inkludieren Sie die `StreamParser.h` in Ihrem Projekt über `#include "StreamParser.h"`.

(3) Sie können nun die StreamParser-DLL-Funktionen verwenden. Siehe auch Kapitel 9 "Beispiel-Code (C++)", Seite 53.

3.2 StreamParser-DLL-Anwenderprogramm entwickeln – Grundsätzliches Vorgehen

Siehe Abbildung 2.

(1) Hardware-Voraussetzungen, Seite 9 einhalten.

(2) Software-Voraussetzungen, Seite 9 einhalten.

(3) Installieren Sie die StreamParser-DLL, siehe Kapitel 3.1 "StreamParser-DLL installieren", Seite 18.

(4) Überprüfen Sie die BIOS-ETH-Version der RTC6 Ethernet-Karte. Aktualisieren Sie auf ≥ BIOS-ETH 40 wie im RTC6-Handbuch, Kapitel 16.7.1 "BIOS-ETH upgraden", Seite 918 beschrieben (wenn nötig).

(5) Rufen Sie in Ihrem RTC6-DLL-Anwenderprogramm den RTC6-Befehl `eth_config_waveform_streaming_ctrl( size, flags )` auf, um die Data-Streaming-Schnittstelle zu konfigurieren.

- `size`
  Nutzdaten-Größe in einem einzelnen Data-Streaming-Datenpaket.
- `flags`
  Muss = 1 sein. Anderenfalls wird die Verbindung von der RTC6 Ethernet-Karte zum StreamParser gleich wieder abgebaut werden.

```
// Pseudo-Code RTC6-DLL
eth_config_waveform_streaming_ctrl( size = [256 |
512 |1024 | 2048 | 4096 | 8192 | 16384 | 32768],
flags = 1 );
```

(6) Rufen Sie im RTC6-DLL-Anwenderprogramm (nach dem Listenausführungsstart!) den RTC6-Befehl **set_trigger[*]** mit `Period`(Bit# 30 = 1) auf.
Damit wird die Data-Streaming-Schnittstelle (auf der RTC6 Ethernet-Karte) konfiguriert und eingeschaltet:

- Die RTC6 Ethernet-Karte füllt ihren Messwertspeicher mit den vom i*DRIVE*-Scan-System übertragenen Daten. Deren Takt ist mit **set_trigger[*]** definiert
- Führt intern den RTC6-Befehl **get_waveform** aus

```
// Pseudo-Code RTC6-DLL
// Periode: 10 µs, Signal 1&2: Scan-Kopf-Status x, y
set_trigger[*]( 1 + Bit #30, 20, 21);
```

(7) Berücksichtigen Sie in Ihrem RTC6-DLL-Anwenderprogramm, dass nach **set_trigger[*]** mit `Period`(Bit# 30 = 1)

- Die Aufzeichnung weiterläuft auch wenn keine Liste mehr ausgeführt wird = nicht mehr beim Listen-Ende stoppt
- Die Aufzeichnung selbst nach **stop_execution** weiterläuft
- Die Aufzeichnung – wie gewöhnlich – nur durch einen **set_trigger[*]**(0)-Aufruf gestoppt wird

(8) Berücksichtigen Sie, dass die RTC6 Ethernet-Karte erst mit dem Data-Streaming beginnt, nachdem sich ein Client mit ihr verbunden hat.

Entwickeln Sie ein Anwenderprogramm, das die von der RTC6 Ethernet-Karte gestreamten Datenpakete empfängt, parst und auswertet, siehe Schritt 9 ff..

(9) StreamParser erzeugen.

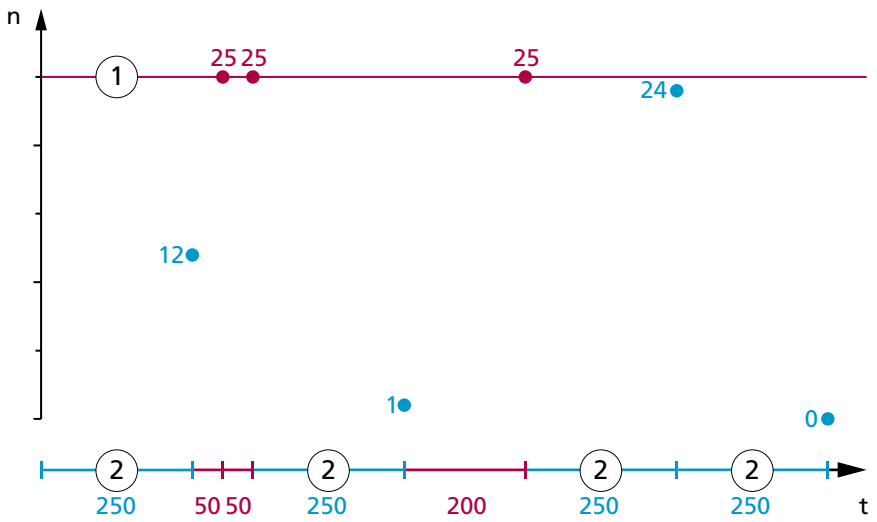
```
// Pseudo-Code
// IP-Adresse,
// Zeiger auf benutzerdefinierte Callback-Funktion,
// deren Kontext
slsp_stream_parser_create(ipaddress, callback,
context, code);
```

Die RTC6 Ethernet-Karte beginnt TCP-Pakete zu streamen.
Die empfangenen Daten werden durch einen impliziten Aufruf der Callback-Funktion abgerufen.

(10) Entwickeln Sie einen Programmteil, der die Daten entsprechend Ihrer Anforderungen weiter verarbeitet (solche Funktionalitäten bietet die StreamParser-DLL nicht).

3.3 Callback-Handler-Aufrufe durch `new_package_counter` und “periodischen Timer”

- Ein Callback-Handler-Aufruf erfolgt:
  - Sobald der “periodische Timer”, Seite 41 abgelaufen ist (t = `new_wait_time_ms`)
  - Sobald n Datenpakete im Stream-Buffer sind (n = `new_package_counter`)
- Folgen eines Callback-Handler-Aufrufs sind:
  - Die momentan vorhandenen Datenpakete im Stream-Buffer werden an den Callback-Handler übergeben = der Stream-Buffer wird geleert
  - Der “periodische Timer”, Seite 41 wird zurückgesetzt
- Zu einer möglichen Abfolge von Callback-Handler-Aufrufen siehe Abbildung 5, Seite 20.



Legende

n	Zum Leeren des Stream-Buffers entnommene Datenpaket-Anzahl
t	[ms]

● (rot) Callback-Handler-Aufruf. Leert den Stream-Buffer. Ursache: n = `new_package_counter` (siehe 1)
● (blau) Callback-Handler-Aufruf. Leert den Stream-Buffer. Ursache: t = `new_wait_time_ms` (siehe 2)

1. Datenpaket-Anzahl im Stream-Buffer, die den Callback-Handler-Aufruf auslöst (n = 25). Definiert durch **slsp_stream_parser_set_package_counter**(`new_package_counter` = 25)
2. Zeitdauer, nach deren Ablauf der Callback-Handler aufgerufen wird und sie erneut beginnt (t = 250 ms). Definiert durch **slsp_stream_parser_set_wait_time_ms**(`new_wait_time_ms` = 250) Wichtig: Auch ● setzt die abgelaufene Zeit wieder auf 0

5

Beispiel: Mögliche Abfolge von Callback-Handler-Aufrufen. Die Rate, mit der die Datenpakete eingehen, variiert in dieser Darstellung.

4 TCP-Paket-Aufbau

Die folgende Tabelle zeigt den Aufbau eines TCP-Pakets, das die RTC6 Ethernet-Karte aussendet, siehe auch Abbildung 2:

- Kopfdaten, Seite 21
- Metadaten, Seite 22
- Nutzdaten, Seite 25

Hinweise

- Sofern nicht anders angegeben, werden die Werte im Little-Endian-Format (= LSB zuerst) übertragen.

Datenpaket-Teil	Größe [Bytes]		Was	Aufbau, Bemerkungen
Kopfdaten	36	2	Protokoll-Version	Aktuell immer 1. Kompatibilitätsinformation
		2	Flags	Bit #31…Bit #0: Reserviert. Beim Aufzeichnen im listenabhängigen Modus, Seite 9 kennzeichnet Bit #0 = 1 das letzte TCP-Paket der Liste
		4	TCP-Paket-Größe	In Bytes.
		8	Zeitstempel	64-Bit-Zeitstempel der RTC6 Ethernet-Karte zum Zeitpunkt der TCP-Paket-Erzeugung Entspricht **get_timestamp_long**
		4	Kanal-Anzahl	Anzahl der Aufzeichnungs-Kanäle
		4	Aufzeichnungs-Periode	`Period`-Wert des letzten set_trigger[*]-Aufrufs
		1	Signal-ID 1	Signal-ID der Werte in Aufzeichnungs-Kanal 1
		1	Signal-ID 2	Signal-ID der Werte in Aufzeichnungs-Kanal 2
		1	Signal-ID 3	Signal-ID der Werte in Aufzeichnungs-Kanal 3
		1	Signal-ID 4	Signal-ID der Werte in Aufzeichnungs-Kanal 4
		1	Signal-ID 5	Signal-ID der Werte in Aufzeichnungs-Kanal 5

Kopfdaten, Seite 21 – Metadaten, Seite 22 – Nutzdaten, Seite 25

Datenpaket-Teil (Forts.)	Größe [Bytes] (Forts.)		Was (Forts.)	Aufbau, Bemerkungen (Forts.)
Kopfdaten (Forts.)	36 (Forts.)	1	Signal-ID 6	Signal-ID der Werte in Aufzeichnungs-Kanal 6
		1	Signal-ID 7	Signal-ID der Werte in Aufzeichnungs-Kanal 7
		1	Signal-ID 8	Signal-ID der Werte in Aufzeichnungs-Kanal 8
		4	Nutzdaten-Größe	Größe der in diesem TCP-Paket enthaltenen Nutzdaten. In Bytes. Entspricht **eth_config_waveform_streaming_ctrl** (`size`)
Metadaten	92	4	Seriennummer der RTC6 Ethernet-Karte	Siehe `SerialNumber`. Entspricht **get_serial_number**.
		4	DSP-Versionsnummer (`RTC6ETH.out`)	Siehe `OUTVersion`. Entspricht **get_hex_version**
		4	FPGA-Versionsnummer und Optionen	Siehe `RBFVersion`. Entspricht **get_rtc_version**
		4	BIOS-Versionsnummer der RTC6 Ethernet-Karte	Siehe `BiosVersion`. Entspricht **get_bios_version**
		4	RTC6 Ethernet-Karten-Status (BUSY-Listenausführungsstatus)	Siehe `ListBusy`. Entspricht **get_status**
		4	Position des Ausgabe-Zeigers	Siehe `OutputPointer`. Entspricht **get_out_pointer**
		4	Anzahl ausgelöster Externer Starts	Siehe `ExtStartCounter`. Entspricht **get_counts**
		4	Anzahl 10 μs-Takt-Überläufe	Siehe `OverrunCounter`. Entspricht **get_overrun**
		4	McBSP-Eingabewert	Siehe `McBSPIn`. Entspricht **get_mcbsp**

Kopfdaten, Seite 21 – Metadaten, Seite 22 – Nutzdaten, Seite 25

Datenpaket-Teil (Forts.)	Größe [Bytes] (Forts.)		Was (Forts.)	Aufbau, Bemerkungen (Forts.)
Metadaten (Forts.)	92 (Forts.)	8	RTC6-Timer-Wert	Siehe `Laptime`. Im 64-Bit-IEEE-754-Format. Entspricht **get_lap_time**
		8	Gespeicherter RTC6-Timer-Wert	Siehe `Timer`. Im 64-Bit-IEEE-754-Format. Entspricht **get_time**
		4	Unterbrechungs-Status	Siehe `WaitStatus`. Entspricht **get_wait_status**
		4	x-Referenzwert (Offsetwert) für die 2D-Encoderkompensation	Siehe `FlyOffsetX`. Entspricht **get_fly_2d_offset**(`OffsetX`)
		4	y-Referenzwert (Offsetwert) für die 2D-Encoderkompensation	Siehe `FlyOffsetY`. Entspricht **get_fly_2d_offset**(`OffsetY`)
		2	Anzahl bisheriger Listenstarts	Untere 4 Bit entsprechen dem Wert in set_trigger[*]-Signal 67 Bit #0...Bit #3
		2	Anzahl bisheriger Listenstopps	Untere 4 Bit entsprechen dem Wert in set_trigger[*]-Signal 67 Bit #4...Bit #7
		4	RTC6-DLL-IP-Adresse	IP-Adresse des PCs, der über die RTC6-DLL verbunden ist, siehe "Haupt"-Anwenderprogramm. Entspricht **eth_get_card_info** Byte 6. *Wichtig*: Big-Endian-Format!
		2	XY2-100-Statuswort des i*DRIVE*-Scan-Systems	Siehe `HeadStatus`. Entspricht **get_head_status**(0)

Kopfdaten, Seite 21 – Metadaten, Seite 22 – Nutzdaten, Seite 25

Datenpaket-Teil (Forts.)	Größe [Bytes] (Forts.)		Was (Forts.)	Aufbau, Bemerkungen (Forts.)
Metadaten (Forts.)	92 (Forts.)	4	"Erweiterter Scan-Kopf-Status" AX	Über "Erweiterter Scan-Kopf-Status" empfangene Werte für i*DRIVE*-Scan-System A, Achse x. Bit #31…Bit #28: Reserviert. Bit #27…Bit #20: "Erweiterter Scan-Kopf-Status"-Index. Bit #19…Bit #0: "Erweiterter Scan-Kopf-Status"-Wert. In jedem Data-Streaming-Datenpaket wird 1 Wert übertragen. Demnach werden 256 TCP-Pakete benötigt, um den "vollen Satz" "Erweiterter Scan-Kopf-Status"-Werte zu erhalten.
		4	"Erweiterter Scan-Kopf-Status" AY	Wie "Erweiterter Scan-Kopf-Status" AX, aber i*DRIVE*-Scan-System A, Achse y.
		4	"Erweiterter Scan-Kopf-Status" BX	Wie "Erweiterter Scan-Kopf-Status" AX, aber i*DRIVE*-Scan-System B, Achse x.
		4	"Erweiterter Scan-Kopf-Status" BY	Wie "Erweiterter Scan-Kopf-Status" AX, aber i*DRIVE*-Scan-System B, Achse y.

Kopfdaten, Seite 21 – Metadaten, Seite 22 – Nutzdaten, Seite 25

Datenpaket-Teil (Forts.)	Größe [Bytes] (Forts.)		Was (Forts.)	Aufbau, Bemerkungen (Forts.)
Nutzdaten	Variabel, entsprechend Nutzdaten-Größe in Kopfdaten	Nutzdaten-Größe / Kanal-Anzahl	Aufzeichnungs-Kanal 1	4 Byte / Wert. Nur vorhanden, wenn mehr als 1 Kanal aufgezeichnet wird
		Nutzdaten-Größe / Kanal-Anzahl	Aufzeichnungs-Kanal 2	4 Byte / Wert. Nur vorhanden, wenn mehr als 1 Kanal aufgezeichnet wird
		Nutzdaten-Größe / Kanal-Anzahl	Aufzeichnungs-Kanal 3	4 Byte / Wert. Nur vorhanden, wenn mehr als 2 Kanäle aufgezeichnet werden
		Nutzdaten-Größe / Kanal-Anzahl	Aufzeichnungs-Kanal 4	4 Byte / Wert. Nur vorhanden, wenn mehr als 2 Kanäle aufgezeichnet werden
		Nutzdaten-Größe / Kanal-Anzahl	Aufzeichnungs-Kanal 5	4 Byte / Wert. Nur vorhanden, wenn mehr als 4 Kanäle aufgezeichnet werden
		Nutzdaten-Größe / Kanal-Anzahl	Aufzeichnungs-Kanal 6	4 Byte / Wert. Nur vorhanden, wenn mehr als 4 Kanäle aufgezeichnet werden
		Nutzdaten-Größe / Kanal-Anzahl	Aufzeichnungs-Kanal 7	4 Byte / Wert. Nur vorhanden, wenn mehr als 4 Kanäle aufgezeichnet werden
		Nutzdaten-Größe / Kanal-Anzahl	Aufzeichnungs-Kanal 8	4 Byte / Wert. Nur vorhanden, wenn mehr als 4 Kanäle aufgezeichnet werden

Kopfdaten, Seite 21 – Metadaten, Seite 22 – Nutzdaten, Seite 25

5 In der API bereitgestellte Funktionen

5.1 Funktions-Übersicht

StreamParser-DLL-Funktionen

StreamParser-bezogen

Anlegen
slsp_stream_parser_create

Zustand abfragen
slsp_stream_parser_get_state

Löschen
slsp_stream_parser_delete

Verbinden mit TCP-Server
slsp_stream_parser_connect

Trennen vom TCP-Server
slsp_stream_parser_disconnect

Besteht Verbindung mit TCP-Server abfragen
slsp_stream_parser_is_connected

Fehler im Extra-Thread der zugehörigen Data-Streaming-Sitzung abfragen
slsp_stream_parser_get_async_error

TCP-Verbindungs-Timeout-Wert setzen
slsp_stream_parser_set_tcp_timeout

Callback-Handler-Aufruf-bezogen

slsp_stream_parser_set_wait_time_ms
Intervall für den "periodischen Timer" ändern

slsp_stream_parser_set_package_counter
Datenpaket-Anzahl im Stream-Buffer einstellen, die einen Callback-Handler-Aufruf auslösen

StreamParser-DLL-bezogen

Version abfragen
slsp_get_version_info

Datenstrom-bezogen

Pop
slsp_rtc_data_stream_pop

Größe feststellen
slsp_rtc_data_stream_get_size

Datenpaket-bezogen

Kanal-Anzahl abfragen
slsp_rtc_data_package_get_channel_count

Metadaten abfragen
slsp_rtc_data_package_get_meta_data

Größe abfragen
slsp_rtc_data_package_get_size

Waveform-Daten abfragen
slsp_rtc_data_package_get_waveform

Waveform-Daten-Typ abfragen
slsp_rtc_data_package_get_waveform_type

Löschen
slsp_rtc_data_package_delete

6

Grafischer Überblick. Siehe auch Kapitel 9 "Beispiel-Code (C++)", Seite 53.

5.2 Funktionsreferenz

In diesem Kapitel:

5.2.1 Allgemeiner Aufbau der Referenztabellen

Name der Funktion	**präfix_name** StreamParser-DLL-Funktionen haben das Präfix "slsp_".
Zweck	Kurze Beschreibung wozu die Funktion geschrieben wurde.

Funktions-Signatur	`datatype präfix name(datatype A, datatype* B, datatype C)`
	`| | | | > Zeile Argument(e) C`
	`| | | > Zeile Argument(e) B`[(a)]
	`| | > Zeile Argument(e) A`
	`| > Zeilen Name der Funktion, Zweck`
	`> Zeile Rückgabewert`

Argument(e)	`A` Datentyp. Kurztext.
	`B` Datentyp. Kurztext.
	`C` Datentyp. Kurztext.
Rückgabewert	Verweis auf eine Beschreibung des Rückgabewerts, z.B. "Siehe Fehlercode, der von den StreamParser-DLL-Funktionen zurückgegeben wird, Seite 50".
Kommentar(e)	• Weitere Informationen zu dieser und verwandten Funktionen. • Verweise auf andere Kapitel und Dokumente.
Code-Beispiel	`// Code-Schnipsel, nicht kompilierbar`
Versionsinfo	Nennt die Version der StreamParser-DLL, in der die Funktion erstmalig zur Verfügung steht, sowie ggf. auch weitere Informationen zu Änderungen.
Verweise	Links zu verwandten Funktionen: **präfix_name_2**

(a) 'datatype*' (Adressoperator) kennzeichnet einen Zeiger.

5.2.2 Datentypen der StreamParser-DLL-Funktionen

Datentyp	Datenformat
`bool`	Bool-Wert • `true` • `false`
`char`	Ein darstellbares Zeichen (character) aus 1 Byte = 8 Bit.
`char*`	Zeiger auf einen \0-terminierten ANSI-String, 1 Byte pro Zeichen. 4 Byte für Win32 Executables. 8 Byte für Win64 Executables. Synonym: Char-Array, C-String.
`double`	64-Bit IEEE Gleitkommaformat. Siehe https://de.wikipedia.org/wiki/IEEE_754.
`double*`	Zeiger auf einen `double`-Wert. `double*` kann auch ein Array sein.
`int32_t`	32-Bit-Wert mit Vorzeichen: $[-2^{31}…+(2^{31}-1)]$.
`int32_t*`	Zeiger auf einen 32-Bit-Wert mit Vorzeichen: $[-2^{31}…+(2^{31}-1)]$. Kann auch ein Zeiger auf ein `int`-Array sein.
`size_t`	Wie definiert in der `stddef.h`.
`size_t*`	Zeiger auf einen `size_t`-Wert. Kann auch ein Zeiger auf ein `size_t`-Array sein.
`uint16_t`	16-Bit-Wert ohne Vorzeichen: $[0…+(2^{16}-1)]$.
`uint32_t`	32-Bit-Wert ohne Vorzeichen: $[0…+(2^{32}-1)]$.
`uint32_t*`	Zeiger auf einen 32-Bit-Wert ohne Vorzeichen: $[0…+(2^{32}-1)]$. Kann auch ein Zeiger auf ein `size_t`-Array sein.
`uint64_t`	64-Bit-Wert ohne Vorzeichen: $[0…+(2^{64}-1)]$.
`uint64_t*`	Zeiger auf einen 64-Bit-Wert ohne Vorzeichen: $[0…+(2^{64}-1)]$. Kann auch ein Zeiger auf ein `size_t`-Array sein.

Hinweise(1)

- `**`
  Zeiger auf einen Zeiger
- `const`
  Der nachfolgend stehende Wert ist nicht veränderbar.
  `const` wird verwendet um diese Werte zu Parameterrückgaben abzugrenzen
- `enum`
  Siehe Kapitel 8 “Aufzählungstypen enum”, Seite 50
- `struct`
  Siehe Kapitel 7 “Strukturen struct”, Seite 43
- `typedef`
  Schlüsselwort, das zur Erstellung eines Alias für einen Datentyp verwendet wird
- `void`
  Funktion liefert keinen Rückgabewert
- `void*`
  Zeiger auf einen generischen Datentyp

(1) Siehe auch https://en.cppreference.com/w/c.

5.2.3 Referenztabellen

Die Abfolge der Referenztabellen in diesem Kapitel ist alphabetisch.

Name der Funktion	**slsp_get_version_info**
Zweck	Gibt Versionsinformationen über die StreamParser-DLL zurück.
Funktions-Signatur	`slsp_version_info slsp_get_version_info(void)`
Argument(e)	-
Rückgabewert	Siehe Struktur `slsp_version_info`.
Kommentar(e)	• –
Code-Beispiel	–
Versionsinfo	Verfügbar ab StreamParser-DLL Vn.n.n.
Verweise	–

Name der Funktion	**slsp_rtc_data_package_delete**
Zweck	Zerstört das Datenpaket.
Funktions-Signatur	`void slsp_rtc_data_package_delete(slsp_rtc_data_package* package, slsp_stream_parser_error_code* code);`
Argument(e)	`package` Zeiger auf ein Datenpaket-Objekt. Von **slsp_rtc_data_stream_pop** zurückgegeben.
	`code` Fehlercode, der von den StreamParser-DLL-Funktionen zurückgegeben wird, Seite 50.
Rückgabewert	–
Kommentar(e)	• **slsp_rtc_data_package_delete** muss (am Ende seiner Nutzung) für jedes `slsp_rtc_data_package` aufgerufen werden, das durch **slsp_rtc_data_stream_pop** abgerufen wurde.
Code-Beispiel	–
Versionsinfo	Verfügbar ab StreamParser-DLL V1.0.0.
Verweise	**slsp_rtc_data_stream_pop**

Name der Funktion	**slsp_rtc_data_package_get_channel_count**
Zweck	Gibt die Anzahl der Kanäle zurück, die mit einem bestimmten Datenpaket verbunden sind.
Funktions-Signatur	`uint32_t slsp_rtc_data_package_get_channel_count(const slsp_rtc_data_package* package, slsp_stream_parser_error_code* code);`
Argument(e)	`package` Zeiger auf ein Datenpaket-Objekt. Von slsp_rtc_data_stream_pop zurückgegeben.
	`code` Fehlercode, der von den StreamParser-DLL-Funktionen zurückgegeben wird, Seite 50.
Rückgabewert	Anzahl der Kanäle eines bestimmten `slsp_rtc_data_package`-Objekts.
Kommentar(e)	• Die Anzahl der Kanäle hängt von für die RTC6 Ethernet-Karte vorgenommenen Einstellungen ab, insbesondere vom genauen **set_trigger[*]**-Aufruf.
Code-Beispiel	–
Versionsinfo	Verfügbar ab StreamParser-DLL V1.0.0.
Verweise	–

Name der Funktion	slsp_rtc_data_package_get_meta_data	
Zweck	Gibt die mit einem bestimmten Datenpaket verbundenen Metadaten zurück.	
Funktions-Signatur	`slsp_rtc_meta_data slsp_rtc_data_package_get_meta_data(const slsp_rtc_data_package* package, slsp_stream_parser_error_code* code;`	
Argument(e)	`package`	Zeiger auf ein Datenpaket-Objekt. Von slsp_rtc_data_stream_pop zurückgegeben.
	`code`	Fehlercode, der von den StreamParser-DLL-Funktionen zurückgegeben wird, Seite 50.
Rückgabewert	Metadaten des Datenpakets. Siehe `struct slsp_rtc_meta_data`.	
Kommentar(e)	• Die Metadaten werden als Teil jedes RTC6 Ethernet-Karten-Datenpakets gesendet. Deren Aktualisierungshäufigkeit hängt daher ab von den für die RTC6 Ethernet-Karte vorgenommenen Einstellungen.	
Code-Beispiel	–	
Versionsinfo	Verfügbar ab StreamParser-DLL V1.0.0.	
Verweise	–	

Name der Funktion	slsp_rtc_data_package_get_size	
Zweck	Gibt die Größe der Waveform-Daten in einem bestimmten Datenpaket zurück.	
Funktions-Signatur	`size_t slsp_rtc_data_package_get_size(const slsp_rtc_data_package* package, slsp_stream_parser_error_code* code);`	
Argument(e)	`package`	Zeiger auf ein Datenpaket-Objekt. Von slsp_rtc_data_stream_pop zurückgegeben.
	`code`	Fehlercode, der von den StreamParser-DLL-Funktionen zurückgegeben wird, Seite 50.
Rückgabewert	Größe der Waveform-Daten.	
Kommentar(e)	• Die Größe wird bestimmt durch: – Die Einstellungen in **n_eth_config_waveform_streaming_ctrl** (im RTC6-DLL-Anwenderprogramm) – Der Anzahl der mit **set_trigger[*]** eingestellten Aufzeichnungs-Kanäle (im RTC6-DLL-Anwenderprogramm)	
Code-Beispiel	–	
Versionsinfo	Verfügbar ab StreamParser-DLL V1.0.0.	
Verweise	**slsp_rtc_data_package_get_waveform**	

Name der Funktion	slsp_rtc_data_package_get_waveform	
Zweck	Kopiert die Waveform-Daten eines Kanals in ein vom Benutzer bereitgestelltes Array.	
Funktions-Signatur	`size_t slsp_rtc_data_package_get_waveform(const slsp_rtc_data_package* package, int32_t* waveform_data, size_t waveform_data_size, uint32_t channel, slsp_stream_parser_error_code* code);`	
Argument(e)	`package`	Zeiger auf ein Datenpaket-Objekt. Von slsp_rtc_data_stream_pop zurückgegeben.
	`waveform_data`	Benutzer müssen einen Array der Größe **slsp_rtc_data_stream_get_size** bereitstellen.
	`waveform_data_size`	Größe der Waveform-Daten.
	`channel`	Nummer des Kanals, bei dem das Signal abgefragt werden soll.
	`code`	Fehlercode, der von den StreamParser-DLL-Funktionen zurückgegeben wird, Seite 50.
Rückgabewert	Größe der kopierten Werte.	
Kommentar(e)	• –	
Code-Beispiel	–	
Versionsinfo	Verfügbar ab StreamParser-DLL V1.0.0.	
Verweise	**slsp_rtc_data_package_get_size**	

Name der Funktion	slsp_rtc_data_package_get_waveform_type	
Zweck	Gibt den mit **set_trigger[*]** eingestellten Kanal-Typ zurück.	
Funktions-Signatur	`uint32_t slsp_rtc_data_package_get_waveform_type(const slsp_rtc_data_package* package, size_t Channel, slsp_stream_parser_error_code* code);`	
Argument(e)	`package`	Zeiger auf ein Datenpaket-Objekt. Von slsp_rtc_data_stream_pop zurückgegeben.
	`Channel`	Nummer zum Abfragen des Typs. Muss kleiner sein als der von **slsp_rtc_data_package_get_channel_count** zurückgegebene Wert.
	`code`	Fehlercode, der von den StreamParser-DLL-Funktionen zurückgegeben wird, Seite 50.
Rückgabewert	Kanal-Typ, der mit **set_trigger[*]** eingestellt wurde.	
Kommentar(e)	• –	
Code-Beispiel	–	
Versionsinfo	Verfügbar ab StreamParser-DLL V1.0.0.	
Verweise	–	

Name der Funktion	slsp_rtc_data_stream_get_size	
Zweck	Gibt die Anzahl der verfügbaren Datenpakete zurück.	
Funktions-Signatur	`size_t slsp_rtc_data_stream_get_size(const slsp_rtc_data_stream* parser, slsp_stream_parser_error_code* code);`	
Argument(e)	`parser`	Zeiger auf ein Datenstrom-Objekt (das von einer benutzerdefinierten Funktion z.B. “MyCallback” geliefert wird).
	`code`	Fehlercode, der von den StreamParser-DLL-Funktionen zurückgegeben wird, Seite 50.
Rückgabewert	Anzahl der verfügbaren Datenpakete. Verringert sich um 1 mit jedem Aufruf von **slsp_rtc_data_stream_pop**.	
Kommentar(e)	• –	
Code-Beispiel	–	
Versionsinfo	Verfügbar ab StreamParser-DLL V1.0.0.	
Verweise	**slsp_rtc_data_stream_pop**	

Name der Funktion	slsp_rtc_data_stream_pop	
Zweck	Holt ein Datenpaket aus dem Datenstrom ab.	
Funktions-Signatur	`slsp_rtc_data_package* slsp_rtc_data_stream_pop(slsp_rtc_data_stream* stream, slsp_stream_parser_error_code* code);`	
Argument(e)	`stream`	Zeiger auf ein Datenstrom-Objekt (das von einer benutzerdefinierten Funktion z.B. “MyCallback” geliefert wird).
	`code`	Fehlercode, der von den StreamParser-DLL-Funktionen zurückgegeben wird, Seite 50.
Rückgabewert	Zeiger auf den kopierten Datenstrom.	
Kommentar(e)	• Der Besitz von `slsp_rtc_data_package` wird an den Aufrufer übertragen. • slsp_rtc_data_package_delete muss (am Ende seiner Nutzung) für jedes `slsp_rtc_data_package` aufgerufen werden, das durch slsp_rtc_data_stream_pop abgerufen wurde.	
Code-Beispiel	–	
Versionsinfo	Verfügbar ab StreamParser-DLL V1.0.0.	
Verweise	**slsp_rtc_data_package_delete**, **slsp_rtc_data_stream_get_size**	

Name der Funktion	slsp_stream_parser_connect	
Zweck	Stellt eine Verbindung zum TCP-Server (RTC6 Ethernet-Karte) her und startet somit die Data-Streaming-Session.	
Funktions-Signatur	`void slsp_stream_parser_connect(const slsp_stream_parser* parser, slsp_stream_parser_error_code* code);`	
Argument(e)	`parser`	Zeiger auf einen StreamParser. Wird von **slsp_stream_parser_create** zurückgegeben.
	`code`	Fehlercode, der von den StreamParser-DLL-Funktionen zurückgegeben wird, Seite 50.
Rückgabewert	Keine.	
Kommentar(e)	• Stellen Sie sicher, dass **eth_config_waveform_streaming_ctrl**c im RTC6-DLL-Anwenderprogramm aufgerufen wurde. Anderenfalls wird der Fehler `slsp_error_code_RECEIVE_PACKAGE_FAILED` zurückgegeben.	
Code-Beispiel	–	
Versionsinfo	Verfügbar ab StreamParser-DLL V1.0.0.	
Verweise	**slsp_stream_parser_disconnect**, **slsp_stream_parser_set_tcp_timeout**, **slsp_stream_parser_set_package_counter**, **slsp_stream_parser_set_wait_time_ms**	

Name der Funktion	slsp_stream_parser_create	
Zweck	Erzeugt einen neuen StreamParser. Gibt einen Zeiger auf diesen zurück.	
Funktions-Signatur	`slsp_stream_parser* slsp_stream_parser_create(const char* ipaddress, slsp_stream_callback callback, void* context, slsp_stream_parser_error_code* code);`	
Argument(e)	`ipaddress`	IP-Adresse der RTC6 Ethernet-Karte, mit der eine Verbindung hergestellt werden soll.
	`callback`	Zeiger auf eine benutzerdefinierte Callback-Funktion für die Verarbeitung des Datenstroms.
	`context`	Zeiger auf den benutzerdefinierten `context`-Parameter der Callback-Funktion.
	`code`	Fehlercode, der von den StreamParser-DLL-Funktionen zurückgegeben wird, Seite 50.
Rückgabewert	StreamParser. Zeiger. Siehe `struct slsp_stream_parser`.	
Kommentar(e)	• **slsp_stream_parser_create** erzeugt eine neue StreamParser. Diese StreamParser kann über den zurückgegebenen Zeiger `slsp_stream_parser*` angesprochen werden. • Das Data-Streaming beginnt nicht, sobald die StreamParser erstellt ist, sondern erst mit **slsp_stream_parser_connect**. • Der Besitz des Objekts `slsp_stream_parser` wird dem Aufrufer über einen raw c-Zeiger übertragen. • Der Aufrufer muss sicherstellen, dass er das Objekt `slsp_stream_parser` am Ende seiner Verwendung mit **slsp_stream_parser_delete** löscht. • Während der Data-Streaming-Session wird der benutzerdefinierte **slsp_stream_callback** wiederholt aufgerufen. Ein Zeiger auf das Datenpaket wird dann an den **slsp_stream_callback** zur benutzerdefinierten Verarbeitung übergeben. • Es wird empfohlen, nur den Inhalt eines **slsp_rtc_data_package** innerhalb des **slsp_stream_callback** zu kopieren und die Daten in einem eigenen Thread zu verarbeiten.	
Code-Beispiel	–	
Versionsinfo	Verfügbar ab StreamParser-DLL V1.0.0.	
Verweise	**slsp_stream_parser_delete**, **slsp_stream_parser_connect**	

Name der Funktion	slsp_stream_parser_delete	
Zweck	Zerstört den angegebenen StreamParser.	
Funktions-Signatur	`void slsp_stream_parser_delete(slsp_stream_parser* parser, slsp_stream_parser_error_code* code);`	
Argument(e)	`parser`	Zeiger auf einen StreamParser. Wird von slsp_stream_parser_create zurückgegeben.
	`code`	Fehlercode, der von den StreamParser-DLL-Funktionen zurückgegeben wird, Seite 50.
Rückgabewert	Keine.	
Kommentar(e)	• Das Data-Streaming wird bei der Zerstörung des StreamParsers gestoppt, wenn es zuvor nicht ordnungsgemäß durch die RTC6 Ethernet-Karte oder einen Netzwerkkommunikationsfehler gestoppt wurde. • Das Data-Streaming kann auch mit **slsp_stream_parser_disconnect** gestoppt werden.	
Code-Beispiel	–	
Versionsinfo	Verfügbar ab StreamParser-DLL V1.0.0.	
Verweise	**slsp_stream_parser_create**, **slsp_stream_parser_disconnect**	

Name der Funktion	slsp_stream_parser_disconnect	
Zweck	Beendet die Verbindung zum TCP-Server (RTC6 Ethernet-Karte) und beendet somit die Data-Streaming-Session.	
Funktions-Signatur	`void slsp_stream_parser_disconnect(const slsp_stream_parser* parser, slsp_stream_parser_error_code* code);`	
Argument(e)	`parser`	Zeiger auf einen StreamParser. Wird von slsp_stream_parser_create zurückgegeben.
	`code`	Fehlercode, der von den StreamParser-DLL-Funktionen zurückgegeben wird, Seite 50.
Rückgabewert	Keine.	
Kommentar(e)	• –	
Code-Beispiel	–	
Versionsinfo	Verfügbar ab StreamParser-DLL V1.0.0.	
Verweise	**slsp_stream_parser_connect**, **slsp_stream_parser_delete**	

Name der Funktion	slsp_stream_parser_get_async_error
Zweck	Gibt die Fehlermeldung zurück, wenn im Extra-Thread der zugehörigen Data-Streaming-Sitzung ein Fehler aufgetreten ist.
Funktions-Signatur	`void slsp_stream_parser_get_async_error(const slsp_stream_parser* parser, slsp_stream_parser_error_code* code);`
Argument(e)	`parser` Zeiger auf einen StreamParser. Wird von slsp_stream_parser_create zurückgegeben. `code` Fehlercode, der von den StreamParser-DLL-Funktionen zurückgegeben wird, Seite 50.
Rückgabewert	Keine.
Kommentar(e)	• –
Code-Beispiel	–
Versionsinfo	Verfügbar ab StreamParser-DLL V1.1.0.
Verweise	–

Name der Funktion	slsp_stream_parser_get_state
Zweck	Gibt den Zustand des StreamParser zurück.
Funktions-Signatur	`slsp_stream_parser_state slsp_stream_parser_get_state(const slsp_stream_parser* parser, slsp_stream_parser_error_code* code);`
Argument(e)	`parser` Zeiger auf einen StreamParser. Wird von slsp_stream_parser_create zurückgegeben. `code` Fehlercode, der von den StreamParser-DLL-Funktionen zurückgegeben wird, Seite 50.
Rückgabewert	Zustand des StreamParser. Siehe `enum slsp_stream_parser_state` und Abbildung 7.
Kommentar(e)	• –
Code-Beispiel	–
Versionsinfo	Verfügbar ab StreamParser-DLL V1.0.0.
Verweise	**slsp_stream_parser_create**

Name der Funktion	slsp_stream_parser_is_connected	
Zweck	Prüft, ob die Data-Streaming-Session eine Verbindung zum TCP-Server hat.	
Funktions-Signatur	`bool slsp_stream_parser_is_connected(const slsp_stream_parser* parser, slsp_stream_parser_error_code* code);`	
Argument(e)	`parser`	Zeiger auf einen StreamParser. Wird von slsp_stream_parser_create zurückgegeben.
	`code`	Fehlercode, der von den StreamParser-DLL-Funktionen zurückgegeben wird, Seite 50.
Rückgabewert	Keine.	
Kommentar(e)	• –	
Code-Beispiel	–	
Versionsinfo	Verfügbar ab StreamParser-DLL V1.0.0.	
Verweise	**slsp_stream_parser_connect**, **slsp_stream_parser_set_tcp_timeout**	

Name der Funktion	slsp_stream_parser_set_package_counter	
Zweck	Stellt die Datenpaket-Anzahl im Stream-Buffer ein, die einen Callback-Handler-Aufruf auslösen.	
Funktions-Signatur	`void slsp_stream_parser_set_package_counter(const slsp_stream_parser* parser, uint32_t new_package_counter, slsp_stream_parser_error_code* code);`	
Argument(e)	`parser`	Zeiger auf einen StreamParser. Wird von slsp_stream_parser_create zurückgegeben.
	`new_package_counter`	Anzahl der Datenpakete im Stream-Buffer. Default-Wert: 25. Zulässige Werte: > 0.
	`code`	Fehlercode, der von den StreamParser-DLL-Funktionen zurückgegeben wird, Seite 50.
Rückgabewert	Keine.	
Kommentar(e)	• Sie müssen **slsp_stream_parser_set_package_counter** vor **slsp_stream_parser_connect** aufrufen. Der `new_package_counter`-Wert kann nur vor der StreamParser-Erstellung gesetzt werden. • Der `new_package_counter`-Wert kann nicht zur Laufzeit geändert werden. • Bei `new_package_counter` = 0 wird der Default-Wert genommen. • Sobald die Datenpaket-Anzahl im Stream-Buffer gleich `new_package_counter` ist: – StreamParser ruft den Callback-Handler auf – StreamParser übergibt die Datenpakete an den Callback-Handler, d. h. leert den Stream-Buffer • Siehe auch Kapitel 3.3 "Callback-Handler-Aufrufe durch `new_package_counter` und "periodischen Timer"", Seite 20	
Code-Beispiel	–	
Versionsinfo	Verfügbar ab StreamParser-DLL V1.1.0.	
Verweise	**slsp_stream_parser_connect**, **slsp_stream_parser_set_wait_time_ms**	

Name der Funktion	**slsp_stream_parser_set_tcp_timeout**	
Zweck	Setzt den TCP-Verbindungs-Timeout-Wert.	
Funktions-Signatur	`void slsp_stream_parser_set_tcp_timeout(const slsp_stream_parser* parser, uint32_t Seconds, slsp_stream_parser_error_code* code);`	
Argument(e)	`parser`	Zeiger auf einen StreamParser. Wird von slsp_stream_parser_create zurückgegeben.
	`Seconds`	TCP-Verbindungs-Timeout-Wert. In s.
	`code`	Fehlercode, der von den StreamParser-DLL-Funktionen zurückgegeben wird, Seite 50.
Rückgabewert	Keine.	
Kommentar(e)	• Default-Wert für den TCP-Verbindungs-Timeout: 10 s. • StreamParser setzt `Seconds` = 0 automatisch auf `Seconds` = 1.	
Code-Beispiel	–	
Versionsinfo	Verfügbar ab StreamParser-DLL V1.0.0.	
Verweise	**slsp_stream_parser_is_connected**	

Name der Funktion	**slsp_stream_parser_set_wait_time_ms**	
Zweck	Ändert das Intervall für den “periodischen Timer” (siehe unten).	
Funktions-Signatur	`void slsp_stream_parser_set_wait_time_ms(const slsp_stream_parser* parser, uint32_t new_wait_time_ms, slsp_stream_parser_error_code* code);`	
Argument(e)	`parser`	Zeiger auf einen StreamParser. Wird von slsp_stream_parser_create zurückgegeben.
	`new_wait_time_ms`	Der zeitliche Abstand, mit dem der “periodische Timer” aufgerufen wird. In ms. Default-Wert: 250 ms. Zulässige Werte: > 0.
	`code`	Fehlercode, der von den StreamParser-DLL-Funktionen zurückgegeben wird, Seite 50.
Rückgabewert	Keine.	
Kommentar(e)	• Sie müssen **slsp_stream_parser_set_wait_time_ms** vor **slsp_stream_parser_connect** aufrufen. Der `new_wait_time_ms`-Wert kann nur vor der StreamParser-Erstellung gesetzt werden. • Der `new_wait_time_ms`-Wert kann nicht zur Laufzeit geändert werden. • Bei `new_wait_time_ms` = 0 wird der Default-Wert genommen. • Der “periodische Timer”: – Löst einen Callback-Handler-Aufruf durch den StreamParser aus – egal wie viele Datenpakete im Stream-Buffer sind – StreamParser übergibt die Datenpakete an den Callback-Handler, d.h.leert den Stream-Buffer – Kann verwendet werden, um regelmäßig zu überprüfen, ob der StreamParser noch “am Leben” ist (= Callback-Handler-Aufruf erfolgt, aber Stream-Buffer hat `size() == 0`) – Siehe auch Kapitel 3.3 “Callback-Handler-Aufrufe durch `new_package_counter` und “periodischen Timer””, Seite 20	
Code-Beispiel	–	
Versionsinfo	Verfügbar ab StreamParser-DLL V1.1.0.	
Verweise	**slsp_stream_parser_connect**, **slsp_stream_parser_set_package_counter**	

6 Funktions-Typen

In diesem Kapitel:

- Funktionstyp **slsp_stream_callback**

Name des Funktionstyps	**slsp_stream_callback**
Beschreibung	Dieser Funktionstyp definiert: • Eine Callback-Funktion
Verwendung	Dieser Funktionstyp wird verwendet durch: • **slsp_stream_parser_create**
Syntax	`typedef void (*slsp_stream_callback)(slsp_rtc_data_stream* stream, void* context);`
Argument(e)	`stream` Datenstrom. Zeiger.
	`context` Benutzer-definierter Kontext. Zeiger.
Kommentar(e)	• **slsp_stream_callback** wird regelmäßig aufgerufen. • Der interne Datenstrom wird nach der Rückkehr von **slsp_stream_callback** (auf den `stream` verweist) geleert. • Sie dürfen den Zeiger * `stream` nicht löschen. Verwenden Sie ihn nur innerhalb dieser Callback-Funktion.
Versionsinfo	Verfügbar ab StreamParser-DLL V1.0.0.
Funktions-Typen, Seite 42	

7 Strukturen struct

In diesem Kapitel:

Name der Struktur	**slsp_rtc_data_package**
Beschreibung	Diese Struktur definiert: • Den Proxy (Stellvertreter) für ein Datenpaket.
Verwendung	Diese Struktur wird verwendet durch: • **slsp_rtc_data_package_delete** • **slsp_rtc_data_package_get_channel_count** • **slsp_rtc_data_package_get_meta_data** • **slsp_rtc_data_package_get_size** • **slsp_rtc_data_package_get_waveform** • **slsp_rtc_data_package_get_waveform_type** • **slsp_rtc_data_stream_get_size** • **slsp_rtc_data_stream_pop**
Syntax	`typedef struct slsp_rtc_data_package slsp_rtc_data_package;`
Argument(e)	`slsp_rtc_data_package` Proxy (Stellvertreter) für ein Datenpaket.
Kommentar(e)	• **slsp_rtc_data_package** wird als Eingabe für die **slsp_data_package[*]**-Funktionen verwendet.
Versionsinfo	Verfügbar ab StreamParser-DLL V1.0.0.
Strukturen struct, Seite 43	

Name der Struktur	**slsp_rtc_data_stream**
Beschreibung	Diese Struktur definiert: • Den Proxy (Stellvertreter) für einen Datenstrom.
Verwendung	Diese Struktur wird verwendet durch: • **slsp_stream_callback** • **slsp_rtc_data_stream_pop** • **slsp_rtc_data_package_get_waveform_type** • **slsp_rtc_data_package_get_size**
Syntax	`typedef struct slsp_rtc_data_stream slsp_rtc_data_stream;`
Argument(e)	`slsp_rtc_data_stream` Proxy (Stellvertreter) für einen Datenstrom.
Kommentar(e)	• **slsp_rtc_data_stream** wird als Eingabe für die **slsp_data_stream[*]**-Funktionen verwendet.
Versionsinfo	Verfügbar ab StreamParser-DLL V1.0.0.
Strukturen struct, Seite 43	

Name der Struktur	slsp_rtc_meta_data		
Beschreibung	Diese Struktur definiert: • Metadaten, Seite 22, die die RTC6 Ethernet-Karte in jedem Datenpaket sendet. • Die Metadaten-Werte stammen von einem zufälligen Zeitpunkt der Datenpaket-Aufzeichnung.		
Verwendung	Diese Struktur wird verwendet durch: • **slsp_rtc_data_package_get_meta_data**		

Syntax	`struct slsp_rtc_meta_data`
	`{`
	`  uint32_t Period;`
	`  uint64_t TimeStamp;`
	`  uint32_t SerialNumber;`
	`  uint32_t OUTVersion;`
	`  uint32_t RBFVersion;`
	`  uint32_t BiosVersion;`
	`  uint32_t ListBusy;`
	`  uint32_t OutputPointer;`
	`  uint32_t ExtStartCounter;`
	`  uint32_t OverrunCounter;`
	`  uint32_t McBSPIn;`
	`  double  Laptime;`
	`  double  Timer;`
	`  uint32_t WaitStatus;`
	`  uint32_t FlyOffsetX;`
	`  uint32_t FlyOffsetY;`
	`  uint16_t ListStartCounter;`
	`  uint16_t ListStopCounter;`
	`  uint32_t MasterIP;`
	`  uint32_t HeadStatus;`
	`};`
	`typedef struct slsp_rtc_meta_data slsp_rtc_meta_data;`		

Argument(e)	`uint32_t`	`Period`	• Aufzeichnungs-Periode • `Period`-Wert des letzten set_trigger[*]-Aufrufs
	`uint64_t`	`TimeStamp`	• Zeitstempel • 64-Bit-Zeitstempel der RTC6 Ethernet-Karte zum Zeitpunkt der TCP-Paket-Erzeugung • Entspricht get_timestamp_long
	`uint32_t`	`SerialNumber`	• Seriennummer der RTC6 Ethernet-Karte • Entspricht get_serial_number.

Name der Struktur	slsp_rtc_meta_data		
Argument(e) (Forts.)	uint32_t	OUTVersion	• DSP-Versionsnummer (RTC6ETH.out) • Entspricht get_hex_version
	uint32_t	RBFVersion	• FPGA-Versionsnummer und Optionen • Entspricht get_rtc_version
	uint32_t	BiosVersion	• BIOS-Versionsnummer der RTC6 Ethernet-Karte • Entspricht get_bios_version
	uint32_t	ListBusy	• RTC6 Ethernet-Karten-Status (BUSY-Listenausführungsstatus) • Entspricht get_status
	uint32_t	OutputPointer	• Position des Ausgabe-Zeigers • Entspricht get_out_pointer
	uint32_t	ExtStartCounter	• Anzahl ausgelöster Externer Starts • Entspricht get_counts
	uint32_t	OverrunCounter	• Anzahl 10 µs-Takt-Überläufe • Entspricht get_overrun
	uint32_t	McBSPIn	• McBSP-Eingabewert • Entspricht get_mcbsp
	double	Laptime	• RTC6-Timer-Wert • Entspricht get_lap_time
	double	Timer	• Gespeicherter RTC6-Timer-Wert • Entspricht get_time
	uint32_t	WaitStatus	• Unterbrechungs-Status • Entspricht get_wait_status
	uint32_t	FlyOffsetX	• x-Referenzwert (Offsetwert) für die 2D-Encoderkompensation • Entspricht get_fly_2d_offset(OffsetX)
	uint32_t	FlyOffsetY	• y-Referenzwert (Offsetwert) für die 2D-Encoderkompensation • Entspricht get_fly_2d_offset(OffsetY)

Name der Struktur	slsp_rtc_meta_data		
Argument(e) (Forts.)	uint16_t	ListStartCounter	• Anzahl bisheriger Listenstarts • Untere 4 Bit entsprechen dem Wert in set_trigger[*]-Signal 67 Bit #0...Bit #3
	uint16_t	ListStopCounter	• Anzahl bisheriger Listenstopps • Untere 4 Bit entsprechen dem Wert in set_trigger[*]-Signal 67 Bit #4...Bit #7
	uint32_t	MasterIP	• RTC6-DLL-IP-Adresse • IP-Adresse des PCs, der über die RTC6-DLL verbunden ist, siehe “Haupt”-Anwenderprogramm. Entspricht eth_get_card_info Byte 6. Wichtig: Big-Endian-Format!
	uint32_t	HeadStatus	• XY2-100-Statuswort des iDRIVE-Scan-Systems • Entspricht get_head_status(0)
Kommentar(e)	• Die Metadaten werden als Teil jedes RTC6 Ethernet-Karten-Datenpakets gesendet. Deren Aktualisierungshäufigkeit hängt daher ab von den für die RTC6 Ethernet-Karte vorgenommenen Einstellungen..		
Versionsinfo	Verfügbar ab StreamParser-DLL V1.0.0.		
Strukturen struct, Seite 43			

Name der Struktur	slsp_stream_parser
Beschreibung	Diese Struktur definiert: • Proxy (Stellvertreter) für die interne StreamParser.
Verwendung	Diese Struktur wird verwendet durch: • **slsp_stream_parser_connect** • **slsp_stream_parser_create** • **slsp_stream_parser_delete** • **slsp_stream_parser_disconnect** • **slsp_stream_parser_get_async_error** • **slsp_stream_parser_get_state** • **slsp_stream_parser_is_connected** • **slsp_stream_parser_set_tcp_timeout**
Syntax	`typedef struct slsp_stream_parser slsp_stream_parser;`
Argument(e)	`slsp_stream_parser` Interne StreamParser.
Kommentar(e)	• –
Versionsinfo	Verfügbar ab StreamParser-DLL V1.0.0.
Strukturen struct, Seite 43	

Name der Struktur	slsp_version_info		
Beschreibung	Diese Struktur definiert: • Die 3 Ziffernblöcke der aktuell laufenden StreamParser-DLL-Version (“Version n.n.n”). Siehe https://semver.org/.		
Verwendung	Diese Struktur wird verwendet durch: • **slsp_get_version_info**		

Syntax	`struct slsp_version_info`
	`{`
	`  uint32_t Major;`
	`  uint32_t Minor;`
	`  uint32_t Patch;`
	`};`
	`typedef struct slsp_version_info slsp_version_info;`		

Argument(e)	`uint32_t`	`Major`	Major-Version der StreamParser-DLL.
	`uint32_t`	`Minor`	Minor-Version der StreamParser-DLL.
	`uint32_t`	`Patch`	Revision-Version (= Patch-Version) der StreamParser-DLL.
Kommentar(e)	• –		
Versionsinfo	Verfügbar ab StreamParser-DLL V1.0.0.		
Strukturen struct, Seite 43			

8 Aufzählungstypen enum

In diesem Kapitel:

Name des enum	**slsp_stream_parser_error_code**	
Beschreibung	Dieses enum legt die Auswahlmöglichkeiten fest für: • Fehlercode, der von den StreamParser-DLL-Funktionen zurückgegeben wird	
Verwendung	Dieses enum wird verwendet durch: • Alle StreamParser-DLL-Funktionen außer **slsp_get_version_info**.	

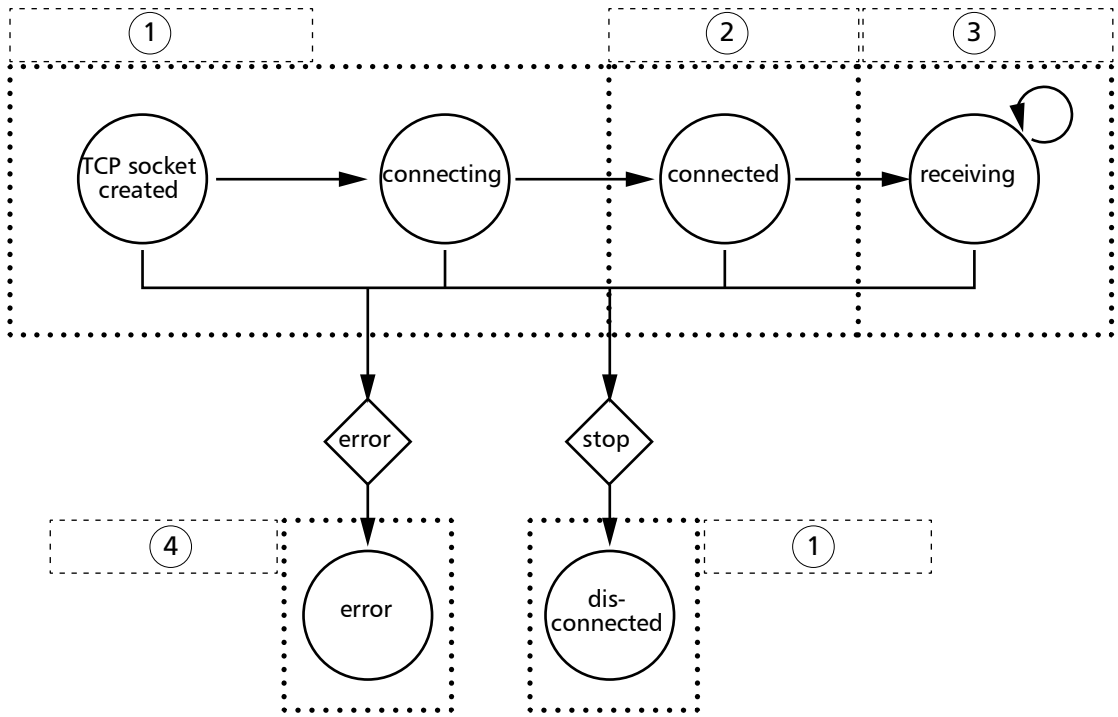
Syntax	`enum slsp_stream_parser_error_code`
	`{`
	`  slsp_error_code_OK = 0,`
	`  slsp_error_code_ESTABLISH_CONNECTION_FAILED = 1,`
	`  slsp_error_code_RECEIVE_PACKAGE_FAILED = 2,`
	`  slsp_error_code_UNKNOWN_ERROR = 3,`
	`  slsp_error_code_RTC_VERSION_ERROR = 4`
	`};`
	`}`
	`typedef enum slsp_stream_parser_error_code slsp_stream_parser_error_code;`	

Aufzählungs-konstante(n)	`slsp_error_code_OK`	Die StreamParser-DLL-Funktion war erfolgreich. Kein Fehler.
	`slsp_error_code_ESTABLISH_CONNECTION_FAILED`	Die StreamParser-DLL-Funktion ist fehlgeschlagen. Die Verbindung zur RTC6 Ethernet-Karte konnte nicht hergestellt werden.
	`slsp_error_code_RECEIVE_PACKAGE_FAILED`	Die StreamParser-DLL-Funktion ist fehlgeschlagen. Es ist ein Fehler aufgetreten beim Versuch, ein Datenpaket von der RTC6 Ethernet-Karte zu empfangen. Siehe auch Seite 35.
	`slsp_error_code_UNKNOWN_ERROR`	Die StreamParser-DLL-Funktion ist fehlgeschlagen. Ein nicht spezifizierter Fehler ist aufgetreten.
	`slsp_error_code_RTC_VERSION_ERROR`	Die RTC6 Ethernet-Karte erfüllt nicht die unter Hardware-Voraussetzungen, Seite 9 und Software-Voraussetzungen, Seite 9 genannten Kriterien.
Versionsinfo	Verfügbar ab StreamParser-DLL V1.1.0.	
Aufzählungstypen enum, Seite 29		

Name des enum	slsp_stream_parser_state	
Beschreibung	Dieses enum legt die Auswahlmöglichkeiten fest für: • Den Zustand des StreamParsers, siehe Abbildung 7.	
Verwendung	Dieses enum wird verwendet durch: • **slsp_stream_parser_get_state**	

Syntax	`enum slsp_stream_parser_state`
	`{`
	`  slsp_STREAM_CONNECTED  = 0,`
	`  slsp_STREAM_RECEIVING  = 1,`
	`  slsp_STREAM_DISCONNECTED  = 2,`
	`  slsp_STREAM_STOPPED_ON_ERROR  = 3`
	`};`
	`typedef enum slsp_stream_parser_state slsp_stream_parser_state;`	

Aufzählungs-konstante(n)	`slsp_STREAM_CONNECTED`	Der TCP-Client ist: • Mit der RTC6 Ethernet-Karte verbunden • Bereit, den Datenstrom zu empfangen Beachten Sie, dass die Callback-Funktion nicht aufgerufen wird.
	`slsp_STREAM_RECEIVING`	Der Datenstrom wird aktuell empfangen: • Die von der RTC6 Ethernet-Karte gesendeten Datenpakete gehen ein • Die Callback-Funktion wird wiederholt (ständig) aufgerufen
	`slsp_STREAM_DISCONNECTED`	Der Datenstrom wurde ordnungsgemäß beendet, weil: • Die RTC6 Ethernet-Karte sendet keine Datenpakete mehr • Es gab eine Trennungsanforderung (**slsp_stream_parser_disconnect**)
	`slsp_STREAM_STOPPED_ON_ERROR`	Der Datenstrom wurde nicht ordnungsgemäß beendet, weil: • Ein Fehler ist aufgetreten
Versionsinfo	Verfügbar ab StreamParser-DLL V1.0.0.	
Aufzählungstypen enum, Seite 29		



Legende

1. StreamParser-Zustand `slsp_STREAM_DISCONNECTED`
2. StreamParser-Zustand `slsp_STREAM_CONNECTED`
3. StreamParser-Zustand `slsp_STREAM_RECEIVING`
4. StreamParser-Zustand `slsp_STREAM_STOPPED_ON_ERROR`

7

StreamParser: Zustandsmaschine.

9 Beispiel-Code (C++)

- Beispiel-Code für RTC6-DLL-Anwenderprogramm
- Beispiel-Code für StreamParser-DLL-Anwenderprogramm

9.1 Beispiel-Code für RTC6-DLL-Anwenderprogramm

```
// Beispiel-Code für RTC6-DLL-Anwenderprogramm
// Aktivieren des Data-Streaming auf RTC6 Ethernet-Karten-Seite
// nicht kompilierbar

uint32_t Size = 256;
n_eth_config_waveform_streaming(Index, Size, 1)

// Listendefinition (set_trigger[*] startet das Data-Streaming)
n_set_start_list(Index, 1);

// Aktiviere Signalaufzeichnung
n_set_trigger4(Index, 1, 1, 2, 20, 21);

// weiterer Code
n_jump_abs(Index, 10, 10);
n_jump_abs(Index, 0,0);
…

// Deaktiviere Signalaufzeichnung (optional, abhängig vom Anwendungsfall)
n_set_trigger4(Index, 0, 1, 2, 20, 21);

// Beende Listendefinition und starte Ausfuehrung der Liste
n_set_end_of_list(Index);
n_execute_list(CardNo, Index);
```

9.2 Beispiel-Code für StreamParser-DLL-Anwenderprogramm

Einen kompilierbaren Beispiel-Code für ein StreamParser-DLL-Anwenderprogramm finden Sie im StreamParser-Softwarepaket unter:

- democode\CPP_Demo.cpp
- democode\C_Demo.cpp

10 Änderungsindex

Nachfolgend genannt sind Änderungen an diesem Handbuch aufgrund technischer Weiterentwicklung des Produkts sowie wesentliche redaktionelle Änderungen.

Dokument-Revision 0.0.9 de-DE

Wo	Was
Global	Dokument-Revision • 0.0.9 de-DE gilt für StreamParser-DLL • V0.5.0.RC3
Global	Vorläufige Version.

Dokument-Revision nach 1.0.0 de-DE von 0.0.9 de-DE

Wo	Was
Global	Dokument-Revision • 1.0.0 de-DE gilt für StreamParser-DLL • V1.0.0
Kapitel 9.2 “Beispiel-Code für StreamParser-DLL-Anwenderprogramm”, Seite 53	Redaktionelle Erweiterung.

Dokument-Revision nach 1.1.0 de-DE von 1.0.0 de-DE

Wo	Was
Global	Dokument-Revision • 1.1.0 de-DE gilt für StreamParser-DLL • V1.1.0
Global	Software-Änderung. Bezeichnung "Erweiterter Scan-Kopf-Status". Umbenannt von "Schmalband-Rückkanal-Multiplexing".
Kapitel 2.4 "Softwarepaket-Inhalt", Seite 11	Software-Änderung.
Kapitel 3 "Softwareentwicklung mit der StreamParser-DLL", Seite 12	Redaktionelle Erweiterung. Abbildung 1, Seite 14. Abbildung 2, Seite 15. Abbildung 3, Seite 16. Abbildung 4, Seite 17.
Kapitel 3.3 "Callback-Handler-Aufrufe durch `new_package_counter` und "periodischen Timer"", Seite 20	Software-Änderung.
slsp_stream_parser_get_async_error, Seite 38	Software-Änderung. Umbenannt von **slsp_stream_parser_get_asnc_error**.
slsp_stream_parser_set_package_counter, Seite 39	Software-Änderung. Neue Funktion.
slsp_stream_parser_set_wait_time_ms, Seite 41	Software-Änderung. Neue Funktion.
slsp_stream_parser_error_code, Seite 50	Software-Änderung. Neue Aufzählungskonstante `slsp_error_code_RTC_VERSION_ERROR`.

Notizen