StreamParser-DLL –
Application Programming Interface

StreamParser DLL V1.1.0

2023-11-23

Doc. Rev. 1.1.0 en-US

SCANLAB GmbH
Siemensstr. 2a
82178 Puchheim
Germany

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

info@scanlab.de
www.scanlab.de

Contents

1 About this Manual

This manual describes the C-API of SCANLAB StreamParser DLL V1.1.0.

⚠ **Caution!**

- Read and observe all safety instructions in this manual!
- SCANLAB accepts no liability for damages or consequential losses resulting from non-observance of this manual, in particular the safety instructions contained herein.

1.1 Manufacturer

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

1.2 Related Documents

- RTC6 Manual

1.3 Glossary

API	Abbreviation of Application Programming Interface. Program part (here: of the StreamParser DLL) which is available for other programs for connecting to the system (here: functions of the StreamParser DLL). See Chapter 5 "Functions Available in the API", page 26.
BIOS	Basic Input/Output System. Is permanently stored in the Flash Memory of the RTC6 board. See also `RTC6conf.exe`.
Callback Event	One of several StreamParser DLL-internal events. See also Callback Function.
Callback Function	Designates a user-supplied function that is to be executed as soon as (StreamParser DLL.-internally) a certain "Callback Event" occurs. See **slsp_stream_callback**, page 42.
Data Packet	Not: TCP Packet, see "TCP Packet Structure", page 21. • Corresponding proxy is `struct slsp_rtc_data_package` • A Data Packet can be queried for: – Number of recording channels – Metadata – Waveform Data of each recording channel
Data Stream	• Corresponding proxy is `struct slsp_rtc_data_stream` • Contains all collected Data Packets since the last call of the Callback Function • Can be queried for number of Data Packets • Each Data Packet can be retrieved
Data Streaming	See Brief information Data Streaming, page 9.
"Extended Scan Head Status"	See RTC6 Manual.
LSB	Least Significant Bit.
Measurement Data Memory (RTC board)	Synonym: waveform memory, ring buffer.
RTC6 Ethernet Board	Refer to RTC6 Manual.
RTC6 DLL User Program	Read: RTC6 DLL-based user program. See also page 15.

StreamParser	Software object - Created by **slsp_stream_parser_create** - Can be addressed by the returned pointer `slsp_stream_parser*`
StreamParser DLL	Generic name for: - `StreamParser.dll` Win32-based dynamic link library - `StreamParser_x64.dll` Win64-based dynamic link library Part of StreamParser Software Package.
StreamParser DLL User Program	Read: StreamParser DLL-based user program. See also page 15.
StreamParser Software Package	See Chapter 2.4 "Software Package Content", page 11.
set_trigger[*]	Read: "**set_trigger**/**set_trigger4**/**set_trigger8**".
User	Designates a person (= "system programmer") who develops StreamParser DLL User Programs.
Waveform Data	The data contained within a recording channel (as set by **set_trigger[*]**). See also Payload, page 25.

2 Product Overview

In this Chapter:

2.1 Intended Use

StreamParser DLL (32-bit version and 64-bit version) is part of the StreamParser Software Package.

For developing user programs under MS Windows it provides a programming interface (API) in the form of functions.

The sole application context of these functions is "Data Streaming" functionality, see Brief information Data Streaming, page 9, with the 2 main use cases:

- "List-dependent mode"
  (Data only from list start to list end)
- "List-independent mode"
  (Data even outside list processing, for example, monitoring temperature)

StreamParser DLL functions allow:

- Establish and disconnect connection to the RTC6 Ethernet Board
- Receiving TCP Packets
- Unpacking (= preparing, parsing) TCP Packets
- Making available TCP Packet contents (after Callback Function call) for further processing

The StreamParser DLL functions can be used in existing program code and thus support the development of custom user programs for:

- Continuous recording (logging) in files or databases
- Visualization
- Non-real-time process control
  (for example, adjust laser power in different additive manufacturing layers)

For access to all functionalities of the StreamParser DLL are available, see Chapter 2.4 "Software Package Content", page 11:

- C-API (documented here)
- C++-API (object-oriented)

⚠ **Caution!**

- Possible personal injury and property damage. Interaction of StreamParser DLL, RTC6 Ethernet Board and Ethernet architecture *does not* meet the requirements for real-time capability.
  Due to the Ethernet-typical latencies in the millisecond range, a reaction (to a data evaluation in the RTC 10 µs clock cycle) is not possible.
  Therefore, do not implement any security features in your user program that are based on StreamParser DLL!
  In particular, you *must not* use the actual position to switch off the laser for safety reasons. Otherwise, there is a risk of possible damage to property or even personal injury.

Brief information Data Streaming

- Hardware prerequisites
  - *iDRIVE* scan system
    - No minimum scan system firmware version required (unlike the Multiplexing functionality)
    - Common data cable
  - RTC6 Ethernet Board
    - ≥ BIOS-ETH 40
    - No certain option required
    - `RTC6ETH.out` ETH 646
    - `RTC6RBF.rbf` RBF 639
- Software prerequisites
  - Windows ≥ 10
  - RTC6 Software Package ≥ V1.16.3
- Data Streaming
  - Improves the possibilities to transfer data
  - Users now have easier access to these data
  - A list can be executing, but it does not necessarily have to be:
    - "List-independent mode"
      - Data even outside list processing, for example, monitoring temperature
      - See Figure 3
    - "List-dependent mode"
      - Data only from list start to list end
      - See Figure 4
  - Is a functionality where the RTC6 Ethernet Board (after an appropriate configuration in the "Main" User Program) independently and continuously transmits data (packed in TCP Packets = Ethernet data packages, not in 10 μs clock cycle, but every n network packets) via a TCP Connection to recipient (for example, StreamParser DLL User Program), see Figure 2. It is a 1:1 connection (read only).
- Content of a TCP Packet:
  - "Header, page 21"
  - "Metadata, page 22"
    - RTC6 Ethernet Board metadata for example, DSP version number (`RTC6ETH.out`)
    - RTC6 Ethernet Board status data for example, Wait state
  - "Payload, page 25"
    = *iDRIVE* scan system status data
    = Data transmitted by the *iDRIVE* scan system to the RTC6 Ethernet Board (configured by **set_trigger[*]**) and then recorded in RTC6 Ethernet Board Measurement Data Memory, see RTC6 Manual, **set_trigger[*]** `Signal1`
- StreamParser
  - Receives TCP Packets
  - Parses them

2.2 Safety

When developing StreamParser DLL User Programs you must observe:

- Safety notice Caution!, page 5
- Safety notice Caution!, page 8

2.3 Prerequisites

- Hardware prerequisites, page 9
- Software prerequisites, page 9

2.4 Software Package Content

```
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 Software Development with StreamParser DLL

In this Chapter:

- "Classic" Procedure (= without Data Streaming) – Pull Mechanism, page 12
- Procedure with Data Streaming and StreamParser DLL – Push Mechanism, page 12

"Classic" Procedure (= without Data Streaming) – Pull Mechanism

If the signals transmitted from the *iDRIVE* scan system to the RTC board – such as its position data – are to be recorded (logged), the following procedure applies (without Data Streaming, without StreamParser DLL):

(1) Create a list that includes a **set_trigger[*]** call.

(2) Start list execution.
As soon as **set_trigger[*]** is called:

The RTC board records the requested data in its Measurement Data Memory

(3) Call **get_waveform**.
Recorded data are transferred from RTC board to PC.

(4) Further processing of the transferred data, however, for the time being "locally" = only in this RTC user program

Important: In the controlling RTC user program, coordination measures (such as monitoring Measurement Data Memory, loading lists) must be programmed as well.

Procedure with Data Streaming and StreamParser DLL – Push Mechanism

- *Important*:
  - Hardware prerequisites, page 9
  - Software prerequisites, page 9

StreamParser DLL is integrated, see also Chapter 3.1 "Installing StreamParser DLL", page 18 to (alternatively):

- The controlling "Main" User Program, see Figure 1, page 14 (A)
- A separate user program (= StreamParser DLL User Program), see Figure 1, page 14 (B)
  - Can run on a separate PC optionally, see Figure 1, page 14 (C)

Procedure:

- See Figure 2, page 15
- See also Example Code for RTC6 DLL User Program
- See also Example Code for StreamParser DLL User Program

(1) Call `eth_config_waveform_streaming_ctrl`.

The RTC6 Ethernet Board Data Streaming interface is switched on and waits for a TCP Client to connect by **slsp_stream_parser_connect** call (that is, is ready to transmit).

(2) Create a list that includes a **set_trigger[*]** call (= specifying desired signals).

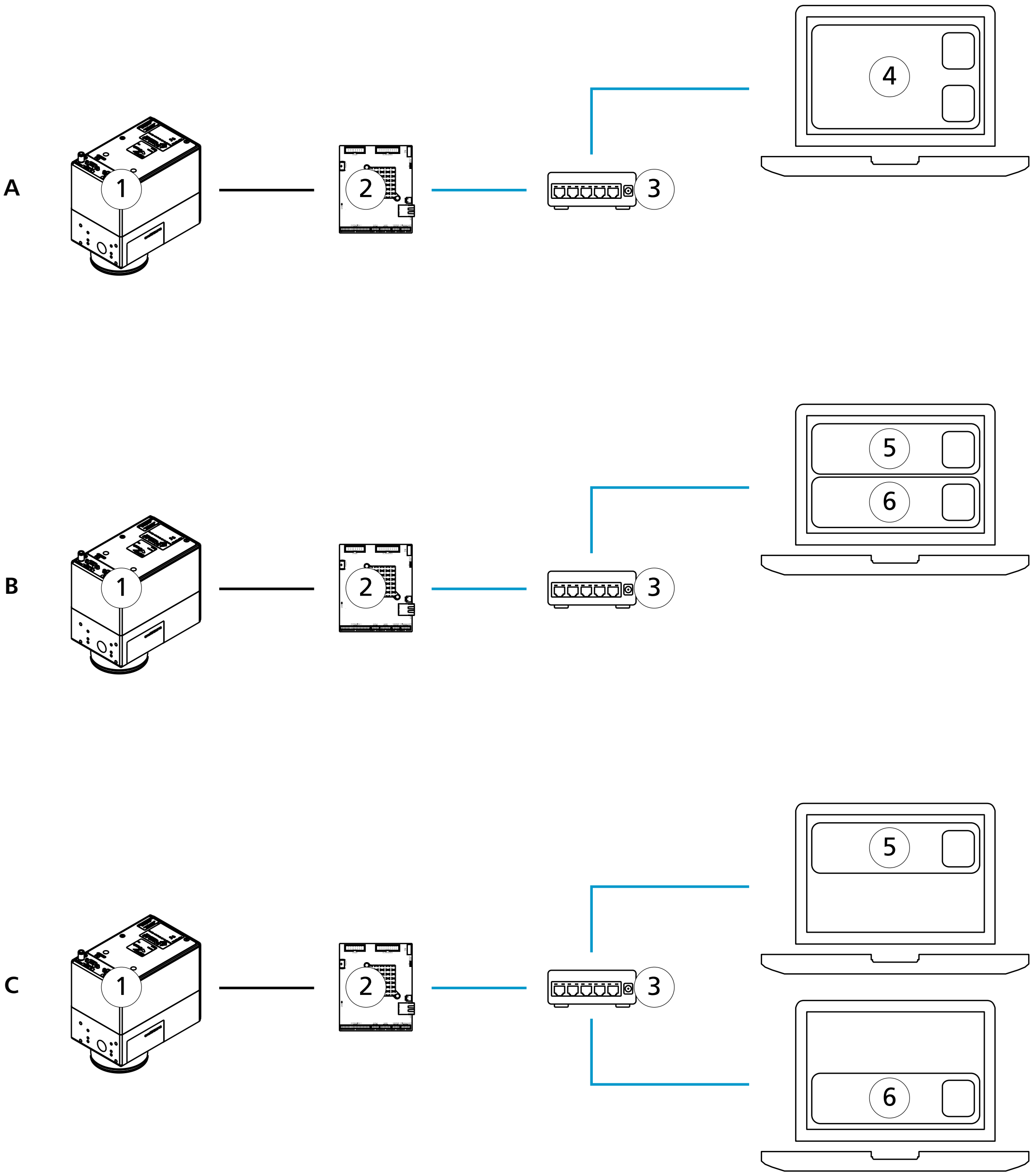
(3) Start list execution.
As soon as **set_trigger[*]** is called:

The RTC6 Ethernet Board continuously transmits the data (packed in TCP Packets) via TCP.

(4) StreamParser DLL receives the TCP Packets, prepares it and makes it available for further processing using a user-implemented Callback Function.

Notes

- *Important*: Further processing (for example, preparing values) is not a functionality of StreamParser DLL.
- No "List-independent mode" mode together with laserDESK protocol functionality.
- Further details can be found in Chapter 3.2 "Developing StreamParser DLL User Programs – General Procedure", page 19.
- For notes and sample code, see Chapter 9 "Example Code (C++)", page 53.



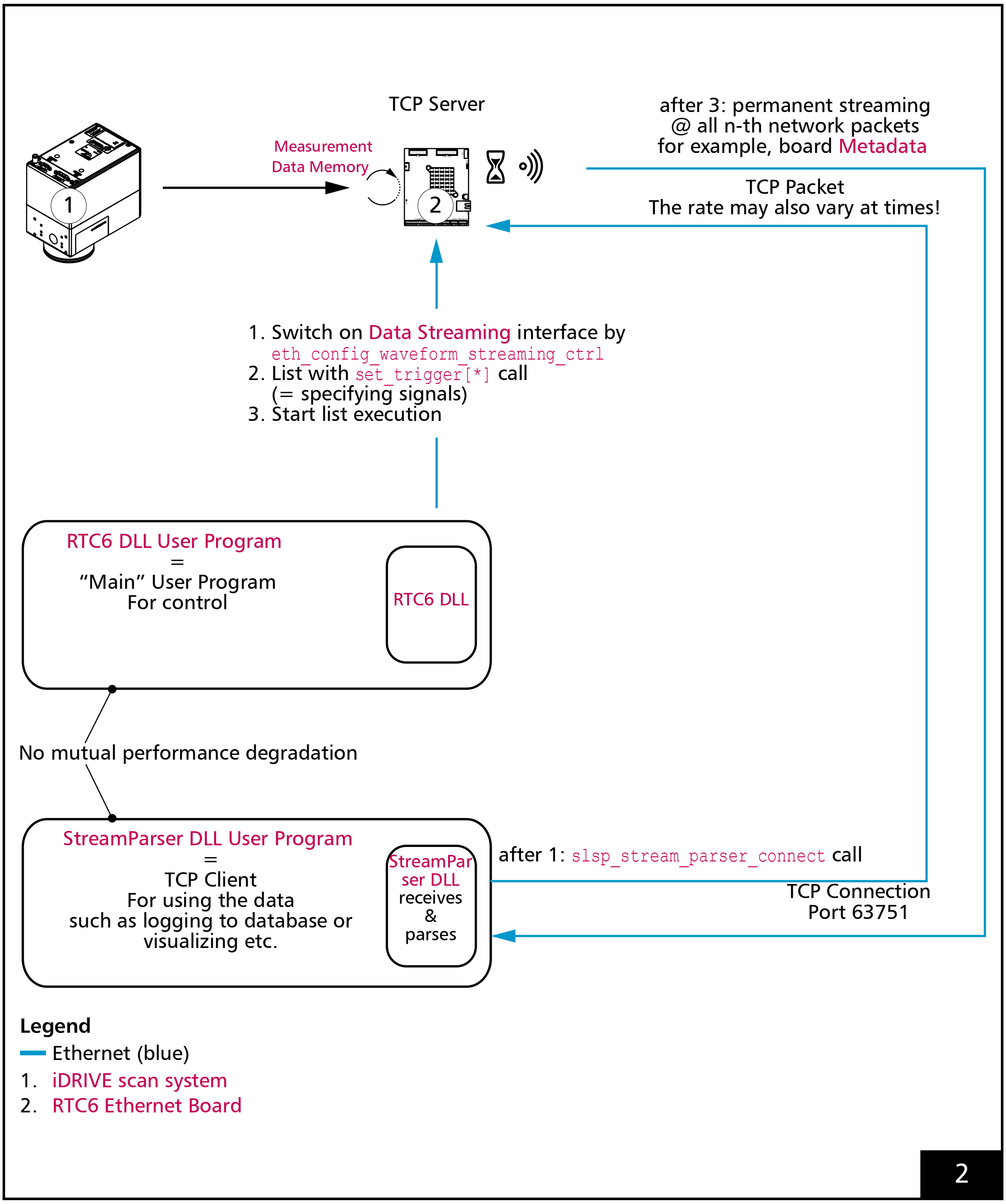
Legend

▬ Ethernet (blue)

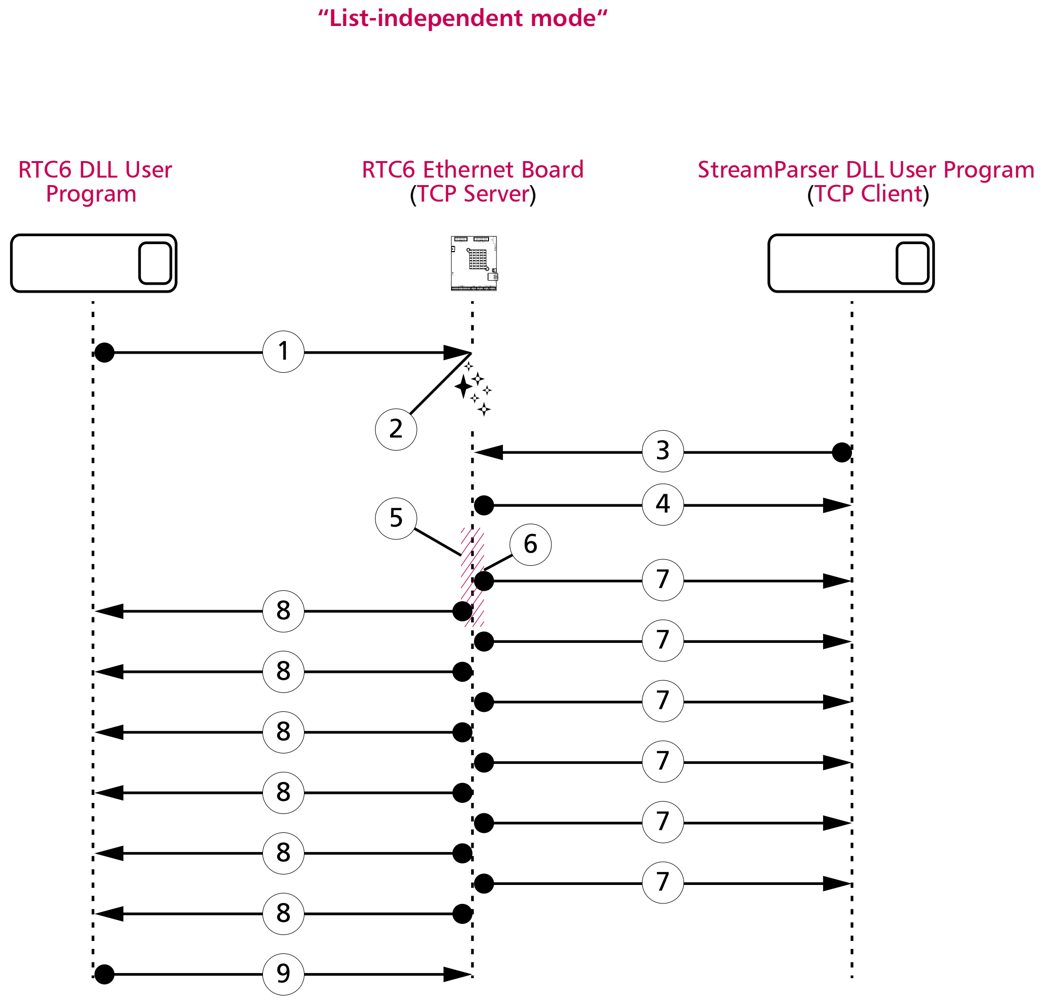
1. iDRIVE scan system
2. RTC6 Ethernet Board
3. Switch, non-optional in (C)
4. 1 user program, basing on RTC6 DLL as well as StreamParser DLL
5. RTC6 DLL User Program
6. StreamParser DLL User Program

1

Topologies. See text, page 12.


TCP Server
after 3: permanent streaming
@ all n-th network packets
for example, board Metadata
Measurement
Data Memory
1
2
TCP Packet
The rate may also vary at times!
1. Switch on Data Streaming interface by
eth_config_waveform_streaming_ctrl
2. List with set_trigger[*] call
(= specifying signals)
3. Start list execution
RTC6 DLL User Program
=
"Main" User Program
For control
RTC6 DLL
No mutual performance degradation
StreamParser DLL User Program
=
TCP Client
For using the data
such as logging to database or
visualizing etc.
StreamParser DLL
receives
&
parses
after 1: slsp_stream_parser_connect call
TCP Connection
Port 63751
Legend
Ethernet (blue)
1. iDRIVE scan system
2. RTC6 Ethernet Board
2

Data Streaming and StreamParser DLL. Scheme.


"List-independent mode"
RTC6 DLL User Program
RTC6 Ethernet Board (TCP Server)
StreamParser DLL User Program (TCP Client)
1
2
3
4
5
6
7
8
9

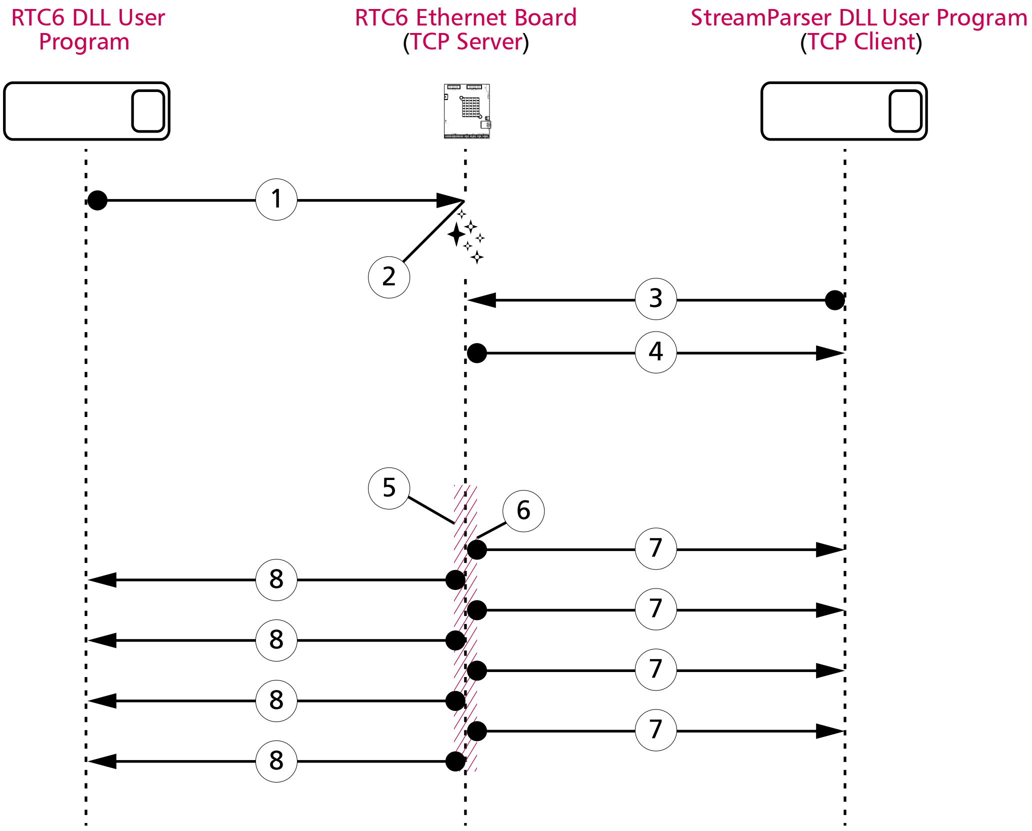
Legend

1. eth_config_waveform_streaming_ctrl call
2. Ready to accept TCP connection
3. TCP Connect
4. TCP Connect OK
5. List, is being executed
6. **set_trigger[*]** call. Triggers the first (7)
   – **set_trigger[*]** with Bit #30 = 1: No automatic recording stop at the end of the list.
   – **set_trigger[*]** with Bit #31 = 1: Endless recording (circular buffer).
7. TCP Packet (according configuration with **set_trigger[*]**)
8. Data Packet
9. **stop_trigger** call. To terminate

3

StreamParser: "List-independent mode". Status diagram.

“List-dependent mode“


RTC6 DLL User Program
RTC6 Ethernet Board (TCP Server)
StreamParser DLL User Program (TCP Client)
1
2
3
4
5
6
7
7
7
7
8
8
8
8

Legend

1. eth_config_waveform_streaming_ctrl call
2. Ready to accept TCP connection
3. TCP Connect
4. TCP Connect OK
5. List, is being executed
6. **set_trigger[*]** call. Triggers the first (7)
   - **set_trigger[*]** with Bit #30 = 0: Automatic recording stop at the end of the list.
   - **set_trigger[*]** with Bit #31 = 1: Endless recording (circular buffer).
7. TCP Packet (according to **set_trigger[*]**)
8. Data Packet

4

StreamParser: “List-dependent mode“. Status diagram.

3.1 Installing StreamParser DLL

There are different ways to integrate the StreamParser DLL in your user program. The following is an example of how to integrate StreamParser DLL into a project.

(1) Add as a dependency to your project:

- `StreamParser.h`
- Alternatively (according to your requirements)
  - For 32-Bit C interface
    - `StreamParser.dll` and `StreamParser.lib`
  - For 64-Bit C interface
    - `StreamParser_x64.dll` and `StreamParser_x64.lib`
  - For 32-Bit C++ interface
    - `StreamParser.dll` and `StreamParser.lib` (release) or `StreamParserd.dll` and `StreamParserd.lib` (debug)
  - For 64-Bit C++ interface
    - `StreamParser_x64.dll` and `StreamParser_x64.lib` (release) or `StreamParser_x64d.dll` and `StreamParser_x64d.lib` (debug)

(2) Include `StreamParser.h` in your project by `#include "StreamParser.h"`.

(3) You can now use the StreamParser DLL functions. See also Chapter 9 "Example Code (C++)", page 53.

3.2 Developing StreamParser DLL User Programs – General Procedure

See Figure 2.

(1) Meet Hardware prerequisites, page 9.

(2) Meet Software prerequisites, page 9.

(3) Install StreamParser DLL,
see Chapter 3.1 "Installing StreamParser DLL", page 18.

(4) Check BIOS-ETH of the RTC6 Ethernet Board.
Upgrade to ≥ BIOS-ETH 40 as described in RTC6 Manual, Chapter 16.7.1 "Upgrading BIOS-ETH", page 918 (if necessary).

(5) In your RTC6 DLL User Program, call RTC6 command `eth_config_waveform_streaming_ctrl( size, flags )` in order to configure the Data Streaming interface.

- `size`
  Payload size in a single Data Streaming Data Packet.
- `flags`
  Must be = 1. Otherwise, the connection from RTC6 Ethernet Board to StreamParser is going to be terminated immediately.

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

(6) In your RTC6 DLL User Program (after list execution start!), call RTC6 command **set_trigger[*]** with `Period`(Bit# 30 = 1).
This configures and turns on Data Streaming interface (on RTC6 Ethernet Board):

- The RTC6 Ethernet Board fills its Measurement Data Memory with the data transmitted by the i*DRIVE* scan system. Their clock cycle is defined by **set_trigger[*]**
- Internally carries out RTC6 command **get_waveform**

```
// Pseudo-Code RTC6 DLL
// Period: 10 µs, Signal 1&2: scan head status x, y
set_trigger[*]( 1 + Bit #30, 20, 21);
```

(7) In your RTC6 DLL User Program, take into account that after **set_trigger[*]** with `Period`(Bit# 30 = 1)

- Recording continues even if no more list is executed = does not stop at the end of the list
- Recording continues even after **stop_execution**
- Recording is stopped – as usual – only by a **set_trigger[*]**(0) call

(8) Take into account that the RTC6 Ethernet Board starts Data Streaming only after a client has been connected to it.

Develop a user program that receives, parses and evaluates the Data Packets streamed from the RTC6 Ethernet Board, see step 9 et seqq..

(9) Create StreamParser.

```
// Pseudo-Code
// IP address,
// Pointer to a user-defined Callback Function,
// its context
slsp_stream_parser_create(ipaddress, callback,
context, code);
```

The RTC6 Ethernet Board starts streaming TCP Packets.
The received data is retrieved by an implicit call of the Callback Function.

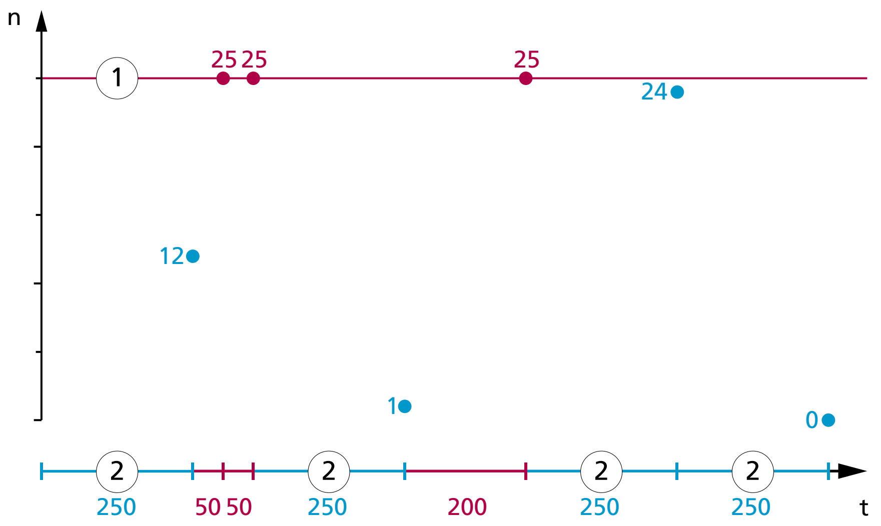
(10) Develop a program part that further processes the data according to your requirements (StreamParser DLL does not offer such functionalities).

3.3 Callback Handler Calls by new_package_counter and "Periodic Timer"

- A callback handler call occurs:
  - As soon as the "periodic timer", page 41 is elapsed (t = new_wait_time_ms)
  - As soon as n Data Packets are in stream buffer (n = new_package_counter)

- Consequences of a callback handler call are:
  - The currently present Data Packets in stream buffer are passed to callback handler = stream buffer is cleared
  - The "periodic timer", page 41 is reset

- For a possible sequence of callback handler calls, see Figure 5, page 20.



Legend

n Data Packet number taken to empty the stream buffer

t [ms]

● (red) Callback handler call. Empties stream buffer. Cause: n = new_package_counter (see 1)

● (blue) Callback handler call. Empties stream buffer. Cause: n = new_package_counter (see 2)

1. Data Packet count in stream buffer that triggers the callback handler call (n = 25). Defined by **slsp_stream_parser_set_package_counter**(new_package_counter = 25)
2. Duration after which the callback handler is called and is restarted (t = 250 ms). Defined by **slsp_stream_parser_set_wait_time_ms**(new_wait_time_ms = 250) Important: ● resets the elapsed time to 0 as well

5

Example: possible sequence of callback handler calls. The rate at which the data Data Packets are received varies in this illustration.

4 TCP Packet Structure

The following table shows the structure of a TCP Packet sent out by the RTC6 Ethernet Board, see also Figure 2:

- Header, page 21
- Metadata, page 22
- Payload, page 25

Notes

- Unless otherwise specified, values are transmitted in Little Endian format (= LSB first).

Data Packet part	Size [Bytes]		What	Structure, Remarks
Header	36	2	Protocol version	To date, always 1. Compatibility information
		2	Flags	Bit #31…Bit #0: Reserved. With recording in **“List-dependent mode“**, page 9 Bit #0 = 1 marks the last TCP Packet of the list
		4	TCP Packet size	In bytes.
		8	Timestamp	64-Bit timestamp made by the RTC6 Ethernet Board upon TCP Packet generation Corresponds to **get_timestamp_long**
		4	Number of channels	Number of recording channels
		4	Recording period	`Period` value of the latest set_trigger[*] call
		1	Signal ID 1	Signal ID of the values in recording channel 1
		1	Signal ID 2	Signal ID of the values in recording channel 2
		1	Signal ID 3	Signal ID of the values in recording channel 3
		1	Signal ID 4	Signal ID of the values in recording channel 4
		1	Signal ID 5	Signal ID of the values in recording channel 5

Header, page 21 – Metadata, page 22 – Payload, page 25

Data Packet part (cont'd.)	Size [Bytes] (cont'd.)		What (cont'd.)	Structure, Remarks (cont'd.)
Header (cont'd)	36 (cont'd)	1	Signal 6	Signal ID of the values in recording channel 6
		1	Signal ID 7	Signal ID of the values in recording channel 7
		1	Signal ID 8	Signal ID of the values in recording channel 8
		4	Payload size	Size of Payload contained in this TCP Packet. In bytes. Corresponds to **eth_config_waveform_streaming_ctrl** (`size`)
Metadata	92	4	Serial number of RTC6 Ethernet Board	See `SerialNumber`. Corresponds to **get_serial_number**.
		4	DSP version number (`RTC6ETH.out`)	See `OUTVersion`. Corresponds to **get_hex_version**
		4	FPGA version number and options	See `RBFVersion`. Corresponds to **get_rtc_version**
		4	BIOS version number of RTC6 Ethernet Board	See `BiosVersion`. Corresponds to **get_bios_version**
		4	RTC6 Ethernet Board status (BUSY list execution status)	See `ListBusy`. Corresponds to **get_status**
		4	Position of output pointer	See `OutputPointer`. Corresponds to **get_out_pointer**
		4	Number of triggered External Starts	See `ExtStartCounter`. Corresponds to **get_counts**
		4	Number of 10 μs clock cycle overruns	See `OverrunCounter`. Corresponds to **get_overrun**
		4	McBSP input value	See `McBSPIn`. Corresponds to **get_mcbsp**

Header, page 21 – Metadata, page 22 – Payload, page 25

Data Packet part (cont'd.)	Size [Bytes] (cont'd.)		What (cont'd.)	Structure, Remarks (cont'd.)
Metadata (cont'd)	92 (cont'd)	8	RTC6 Timer value	See `Laptime`. In 64-Bit-IEEE-754 format. Corresponds to **get_lap_time**
		8	Saved RTC6 Timer value	See `Timer`. In 64-Bit-IEEE-754 format. Corresponds to **get_time**
		4	Wait state	See `WaitStatus`. Corresponds to **get_wait_status**
		4	x reference value (offset value) for 2D encoder compensation	See `FlyOffsetX`. Corresponds to **get_fly_2d_offset**(`OffsetX`)
		4	y reference value (offset value) for 2D encoder compensation	See `FlyOffsetY`. Corresponds to **get_fly_2d_offset**(`OffsetY`)
		2	Number of list starts to date	Lower 4 bits corresponds to the value in set_trigger[*] signal 67 Bit #0...Bit #3
		2	Number of list stops to date	Lower 4 bits corresponds to the value in set_trigger[*] signal 67 Bit #4...Bit #7
		4	RTC6 DLL IP address	IP address of the PC which is connected by RTC6 DLL, see "Main" User Program. Corresponds to **eth_get_card_info** Byte 6. *Important*: Big Endian format!
		2	XY2-100 status word of *iDRIVE* scan system	See `HeadStatus`. Corresponds to **get_head_status**(0)

Header, page 21 – Metadata, page 22 – Payload, page 25

Data Packet part (cont'd.)	Size [Bytes] (cont'd.)		What (cont'd.)	Structure, Remarks (cont'd.)
Metadata (cont'd)	92 (cont'd)	4	"Extended Scan Head Status" AX	Received values by "Extended Scan Head Status" for i*DRIVE* scan system A, axis x. Bit #31…Bit #28: Reserved. Bit #27…Bit #20: "Extended Scan Head Status" index. Bit #19…Bit #0: "Extended Scan Head Status" value. 1 value is transmitted in each Data Streaming Data Packet. Accordingly, 256 TCP Packets are required to obtain the "full set" of "Extended Scan Head Status" values.
		4	"Extended Scan Head Status" AY	Like "Extended Scan Head Status" AX, but i*DRIVE* scan system A, axis y.
		4	"Extended Scan Head Status" BX	Like "Extended Scan Head Status" AX, but i*DRIVE* scan system B, axis x.
		4	"Extended Scan Head Status" BY	Like "Extended Scan Head Status" AX, but i*DRIVE* scan system B, axis y.

Header, page 21 – Metadata, page 22 – Payload, page 25

Data Packet part (cont'd.)	Size [Bytes] (cont'd.)		What (cont'd.)	Structure, Remarks (cont'd.)
Payload	Variable, corresponding to Payload size in Header	Payload size / Number of channels	recording channel 1	4 bytes / value. Only present, if more than 1 channel is recorded
		Payload size / Number of channels	recording channel 2	4 bytes / value. Only present, if more than 1 channel is recorded
		Payload size / Number of channels	recording channel 3	4 bytes / value. Only present, if more than 2 channels are recorded
		Payload size / Number of channels	recording channel 4	4 bytes / value. Only present, if more than 2 channels are recorded
		Payload size / Number of channels	recording channel 5	4 bytes / value. Only present, if more than 4 channels are recorded
		Payload size / Number of channels	recording channel 6	4 bytes / value. Only present, if more than 4 channels are recorded
		Payload size / Number of channels	recording channel 7	4 bytes / value. Only present, if more than 4 channels are recorded
		Payload size / Number of channels	recording channel 8	4 bytes / value. Only present, if more than 4 channels are recorded

Header, page 21 – Metadata, page 22 – Payload, page 25

5 Functions Available in the API

5.1 Function Overview

StreamParser DLL functions

StreamParser-related

To create
slsp_stream_parser_create

To query status
slsp_stream_parser_get_state

To delete
slsp_stream_parser_delete

To connect to TCP Server
slsp_stream_parser_connect

To disconnect from TCP Server
slsp_stream_parser_disconnect

To query whether connection to TCP Server is alive
slsp_stream_parser_is_connected

To query error in extra thread of the associated Data Streaming session
slsp_stream_parser_get_async_error

Set TCP Connection timeout value
slsp_stream_parser_set_tcp_timeout

Callback handler call-related

slsp_stream_parser_set_wait_time_ms
To change the interval for the ”periodic timer”

slsp_stream_parser_set_package_counter
To set the number of Data Packets in stream buffer that trigger a callback handler call

StreamParser DLL-related

To query version
slsp_get_version_info

Data Stream-related

To pop
slsp_rtc_data_stream_pop

To query size
slsp_rtc_data_stream_get_size

Data Packet-related

To query number of channels
slsp_rtc_data_package_get_channel_count

To query Metadata
slsp_rtc_data_package_get_meta_data

To query size
slsp_rtc_data_package_get_size

To query Waveform Data
slsp_rtc_data_package_get_waveform

To query the Waveform Data type
slsp_rtc_data_package_get_waveform_type

To delete
slsp_rtc_data_package_delete

6

Graphical overview. See also Chapter 9 ”Example Code (C++)”, page 53.

5.2 Function Reference

In this Chapter:

- Chapter 5.2.1 ”General Structure of the Reference Tables”, page 27
- Chapter 5.2.2 ”Data Types of the StreamParser DLL Functions”, page 28
- Chapter 5.2.3 ”Reference Tables”, page 30

5.2.1 General Structure of the Reference Tables

Name of the function		**prefix_name** StreamParser DLL functions have the prefix ”slsp_”.
Purpose		Short description describing the purpose of the function.

Function signature		`datatype prefix_name(datatype A, datatype* B, datatype C)`
		`   |         |            |         |          > Line Argument(s) C`
		`   |         |            |          > Line Argument(s) B(a)`
		`   |         |             > Line Argument(s) A`
		`   |          > Lines Name of the function, Purpose`
		`    > Line Return value`

Argument(s)	`A`	Data type. Short text.
	`B`	Data type. Short text.
	`C`	Data type. Short text.
Return value		Reference to a description of the return value, for example, ”See Error code returned by the StreamParser DLL functions, page 50”.
Comment(s)		• Additional information on this and similar functions. • References to other chapters and publications.
Code example		`// Code snippet, not compilable`
Version info		States the StreamParser DLL version in which the function has been published for the first time and, if applicable, further information on changes.
References		Links to related functions: **prefix_name_2**

(a) ’datatype*’ (address operator) indicates a pointer.

5.2.2 Data Types of the StreamParser DLL Functions

Data type	Data format
`bool`	Boolean value • `true` • `false`
`char`	A presentable character of 1 byte = 8 bit.
`char*`	Pointer to a \0-terminated ANSI string, 1 byte per char. 4 Byte for Win32 executables. 8 Byte for Win64 executables. Synonym: char array, C-string.
`double`	64-bit IEEE floating point format. See https://en.wikipedia.org/wiki/IEEE_754.
`double*`	Pointer to a `double` value. `double*` can be an array also.
`int32_t`	Signed 32-bit value: $[-2^{31}\ldots+(2^{31}-1)]$.
`int32_t*`	Pointer to a signed 32-bit value: $[-2^{31}\ldots+(2^{31}-1)]$. Can also be a pointer to an `int` array.
`size_t`	As defined in `stddef.h`.
`size_t*`	Pointer to a `size_t` value. Can also be a pointer to a `size_t` array.
`uint16_t`	Unsigned 16-bit value: $[0\ldots+(2^{16}-1)]$. Synonym: `unsigned short`.
`uint32_t`	Unsigned 32-bit value: $[0\ldots+(2^{32}-1)]$. Synonym: `unsigned int`.
`uint32_t*`	Pointer to a unsigned 32-bit value: $[0\ldots+(2^{32}-1)]$. Can also be a pointer to a `size_t` array.
`uint64_t`	Unsigned 64-bit value: $[0\ldots+(2^{64}-1)]$.
`uint64_t*`	Pointer to a unsigned 64-bit value: $[0\ldots+(2^{64}-1)]$. Can also be a pointer to a `size_t` array.

Notes[1]

- `**`
  pointer to a pointer
- `const`
  the value that follows is not changeable.
  `const` is used to differentiate these values from returned parameter values
- `enum`
  see Chapter 8 "Enumerated Types enum", page 50
- `struct`
  see Chapter 7 "Structures struct", page 43
- `typedef`
  keyword that is used to create an alias for a data type
- `void`
  function does not deliver a return value
- `void*`
  pointer to a generic data type

[1] see also https://en.cppreference.com/w/c.

5.2.3 Reference Tables

The sequence of the reference tables in this chapter is alphabetically.

Name of the function	**slsp_get_version_info**
Purpose	Returns version info on StreamParser DLL.
Function signature	`slsp_version_info slsp_get_version_info(void)`
Argument(s)	–
Return value	See struct `slsp_version_info`.
Comment(s)	• –
Code example	–
Version info	Available as of StreamParser DLL Vn.n.n.
References	–

Name of the function	**slsp_rtc_data_package_delete**	
Purpose	Destroys the Data Packet.	
Function signature	`void slsp_rtc_data_package_delete(slsp_rtc_data_package* package, slsp_stream_parser_error_code* code);`	
Argument(s)	`package`	Pointer to a Data Packet object. Returned by **slsp_rtc_data_stream_pop**.
	`code`	Error code returned by the StreamParser DLL functions, page 50.
Return value	–	
Comment(s)	- **slsp_rtc_data_package_delete** needs to be called (at the end of its usage) for each `slsp_rtc_data_package` that has been retrieved by **slsp_rtc_data_stream_pop**.	
Code example	–	
Version info	Available as of StreamParser DLL V1.0.0.	
References	**slsp_rtc_data_stream_pop**	

Name of the function	**slsp_rtc_data_package_get_channel_count**	
Purpose	Returns the number of channels associated with a particular Data Packet.	
Function signature	`uint32_t slsp_rtc_data_package_get_channel_count(const slsp_rtc_data_package* package, slsp_stream_parser_error_code* code);`	
Argument(s)	`package`	Pointer to a Data Packet object. Returned by slsp_rtc_data_stream_pop.
	`code`	Error code returned by the StreamParser DLL functions, page 50.
Return value	Number of channels of a particular `slsp_rtc_data_package` object.	
Comment(s)	- The number of channels depends on the settings made for the RTC6 Ethernet Board, in particular on the exact **set_trigger[*]** call.	
Code example	–	
Version info	Available as of StreamParser DLL V1.0.0.	
References	–	

Name of the function	**slsp_rtc_data_package_get_meta_data**
Purpose	Returns the Metadata associated with a certain Data Packet.
Function signature	`slsp_rtc_meta_data slsp_rtc_data_package_get_meta_data(const slsp_rtc_data_package* package, slsp_stream_parser_error_code* code);`
Argument(s)	`package` Pointer to a Data Packet object. Returned by slsp_rtc_data_stream_pop. `code` Error code returned by the StreamParser DLL functions, page 50.
Return value	Metadata of a Data Packet. See `struct slsp_rtc_meta_data`.
Comment(s)	• The Metadata are sent with every RTC6 Ethernet Board Data Packet. Therefore, its update frequency depends on the settings made for the RTC6 Ethernet Board.
Code example	–
Version info	Available as of StreamParser DLL V1.0.0.
References	–

Name of the function	**slsp_rtc_data_package_get_size**
Purpose	Returns the size of Waveform Data in a particular Data Packet.
Function signature	`size_t slsp_rtc_data_package_get_size(const slsp_rtc_data_package* package, slsp_stream_parser_error_code* code);`
Argument(s)	`package` Pointer to a Data Packet object. Returned by slsp_rtc_data_stream_pop. `code` Error code returned by the StreamParser DLL functions, page 50.
Return value	Size of Waveform Data.
Comment(s)	• The size is determined by: – The settings in **n_eth_config_waveform_streaming_ctrl** (in RTC6 DLL User Program) – The number of recording channels set by **set_trigger[*]** (in RTC6 DLL User Program)
Code example	–
Version info	Available as of StreamParser DLL V1.0.0.
References	**slsp_rtc_data_package_get_waveform**

Name of the function	slsp_rtc_data_package_get_waveform	
Purpose	Copies the Waveform Data of one channel to a user-supplied array.	
Function signature	`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(s)	`package`	Pointer to a Data Packet object. Returned by slsp_rtc_data_stream_pop.
	`waveform_data`	Users have to supply an array of size **slsp_rtc_data_stream_get_size**.
	`waveform_data_size`	Size of Waveform Data.
	`channel`	Number of the channel where the signal is to be queried.
	`code`	Error code returned by the StreamParser DLL functions, page 50.
Return value	Size of copied values.	
Comment(s)	• –	
Code example	–	
Version info	Available as of StreamParser DLL V1.0.0.	
References	**slsp_rtc_data_package_get_size**	

Name of the function	slsp_rtc_data_package_get_waveform_type	
Purpose	Returns the channel type set by **set_trigger[*]**.	
Function signature	`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(s)	`package`	Pointer to a Data Packet object. Returned by slsp_rtc_data_stream_pop.
	`Channel`	Number to query the type. Must be smaller than the value returned by **slsp_rtc_data_package_get_channel_count**.
	`code`	Error code returned by the StreamParser DLL functions, page 50.
Return value	Channel type set by **set_trigger[*]**.	
Comment(s)	• –	
Code example	–	
Version info	Available as of StreamParser DLL V1.0.0.	
References	–	

Name of the function	slsp_rtc_data_stream_get_size	
Purpose	Returns the number of available Data Packets.	
Function signature	`size_t slsp_rtc_data_stream_get_size(const slsp_rtc_data_stream* parser, slsp_stream_parser_error_code* code);`	
Argument(s)	`parser`	Pointer to a Data Stream object (which is returned by a user-defined function, for example, “MyCallback”).
	`code`	Error code returned by the StreamParser DLL functions, page 50.
Return value	Number of available Data Packets. Decreases by 1 with each call of **slsp_rtc_data_stream_pop**.	
Comment(s)	• –	
Code example	–	
Version info	Available as of StreamParser DLL V1.0.0.	
References	**slsp_rtc_data_stream_pop**	

Name of the function	slsp_rtc_data_stream_pop	
Purpose	Retrieves a Data Packet from Data Stream.	
Function signature	`slsp_rtc_data_package* slsp_rtc_data_stream_pop(slsp_rtc_data_stream* stream, slsp_stream_parser_error_code* code);`	
Argument(s)	`stream`	Pointer to a Data Stream object (which is returned by a user-defined function, for example, “MyCallback”).
	`code`	Error code returned by the StreamParser DLL functions, page 50.
Return value	Pointer to the copied Data Stream.	
Comment(s)	• The ownership of `slsp_rtc_data_package` is transferred to the caller. • slsp_rtc_data_package_delete needs to be called (at the end of its usage) for each `slsp_rtc_data_package` that has been retrieved by slsp_rtc_data_stream_pop.	
Code example	–	
Version info	Available as of StreamParser DLL V1.0.0.	
References	**slsp_rtc_data_package_delete**, **slsp_rtc_data_stream_get_size**	

Name of the function	**slsp_stream_parser_connect**	
Purpose	Establishes a connection to the TCP Server (RTC6 Ethernet Board) and thus starts the Data Streaming session.	
Function signature	`void slsp_stream_parser_connect(const slsp_stream_parser* parser, slsp_stream_parser_error_code* code);`	
Argument(s)	`parser`	Pointer to an StreamParser. Returned by **slsp_stream_parser_create**.
	`code`	Error code returned by the StreamParser DLL functions, page 50.
Return value	None.	
Comment(s)	• Make sure that **eth_config_waveform_streaming_ctrl** has been called in RTC6 DLL User Program. Otherwise, the error `slsp_error_code_RECEIVE_PACKAGE_FAILED` is returned.	
Code example	–	
Version info	Available as of StreamParser DLL V1.0.0.	
References	**slsp_stream_parser_disconnect**, **slsp_stream_parser_set_tcp_timeout**, **slsp_stream_parser_set_package_counter**, **slsp_stream_parser_set_wait_time_ms**	

Name of the function	slsp_stream_parser_create	
Purpose	Creates a new StreamParser. Returns a pointer to it.	
Function signature	`slsp_stream_parser* slsp_stream_parser_create(const char* ipaddress, slsp_stream_callback callback, void* context, slsp_stream_parser_error_code* code);`	
Argument(s)	`ipaddress`	IP address of the RTC6 Ethernet Board to connect to.
	`callback`	Pointer to the user defined Callback Function for processing the Data Stream.
	`context`	Pointer to user defined `context` parameter of the Callback Function.
	`code`	Error code returned by the StreamParser DLL functions, page 50.
Return value	StreamParser. Pointer. See `struct slsp_stream_parser`.	
Comment(s)	• **slsp_stream_parser_create** creates a new StreamParser. This StreamParser can be addressed by the returned pointer `slsp_stream_parser*`. • The Data Streaming does not start as soon as the StreamParser has been created, but with **slsp_stream_parser_connect**. • Ownership of the object `slsp_stream_parser` is transferred to the caller via an raw c pointer. • The caller needs to make sure to delete the object `slsp_stream_parser` at the end of its usage with **slsp_stream_parser_delete**. • During the Data Streaming session, the user defined **slsp_stream_callback** is called repeatedly. A pointer to the Data Packet is then passed to the **slsp_stream_callback** for user defined processing. • It is recommended to only copy the content of a **slsp_rtc_data_package** inside the **slsp_stream_callback** and process the data in an extra thread.	
Code example	–	
Version info	Available as of StreamParser DLL V1.0.0.	
References	**slsp_stream_parser_delete**, **slsp_stream_parser_connect**	

Name of the function	slsp_stream_parser_delete
Purpose	Destroys the specified StreamParser.
Function signature	`void slsp_stream_parser_delete(slsp_stream_parser* parser, slsp_stream_parser_error_code* code);`
Argument(s)	`parser` Pointer to an StreamParser. Returned by slsp_stream_parser_create.
	`code` Error code returned by the StreamParser DLL functions, page 50.
Return value	None.
Comment(s)	• Data Streaming stops on destruction of StreamParser, if it has not been properly stopped by the RTC6 Ethernet Board or a network communication error before. • Data Streaming can also be stopped by **slsp_stream_parser_disconnect**.
Code example	–
Version info	Available as of StreamParser DLL V1.0.0.
References	**slsp_stream_parser_create**, **slsp_stream_parser_disconnect**

Name of the function	slsp_stream_parser_disconnect
Purpose	Terminates the connection to TCP Server (RTC6 Ethernet Board) and thus ends the Data Streaming session.
Function signature	`void slsp_stream_parser_disconnect(const slsp_stream_parser* parser, slsp_stream_parser_error_code* code);`
Argument(s)	`parser` Pointer to an StreamParser. Returned by slsp_stream_parser_create.
	`code` Error code returned by the StreamParser DLL functions, page 50.
Return value	None.
Comment(s)	• –
Code example	–
Version info	Available as of StreamParser DLL V1.0.0.
References	**slsp_stream_parser_connect**, **slsp_stream_parser_delete**

Name of the function	slsp_stream_parser_get_async_error
Purpose	Returns the error message if an error occurred in the extra thread of the associated Data Streaming session.
Function signature	`void slsp_stream_parser_get_async_error(const slsp_stream_parser* parser, slsp_stream_parser_error_code* code);`
Argument(s)	`parser` Pointer to an StreamParser. Returned by slsp_stream_parser_create. `code` Error code returned by the StreamParser DLL functions, page 50.
Return value	None.
Comment(s)	• –
Code example	–
Version info	Available as of StreamParser DLL V1.1.0.
References	–

Name of the function	slsp_stream_parser_get_state
Purpose	Returns the state of StreamParser.
Function signature	`slsp_stream_parser_state slsp_stream_parser_get_state(const slsp_stream_parser* parser, slsp_stream_parser_error_code* code);`
Argument(s)	`parser` Pointer to an StreamParser. Returned by slsp_stream_parser_create. `code` Error code returned by the StreamParser DLL functions, page 50.
Return value	State of StreamParser. See `enum slsp_stream_parser_state` and Figure 7.
Comment(s)	• –
Code example	–
Version info	Available as of StreamParser DLL V1.0.0.
References	slsp_stream_parser_create

Name of the function	slsp_stream_parser_is_connected
Purpose	Checks, if the Data Streaming session has a connection to the TCP Server.
Function signature	`bool slsp_stream_parser_is_connected(const slsp_stream_parser* parser, slsp_stream_parser_error_code* code);`
Argument(s)	`parser` Pointer to an StreamParser. Returned by slsp_stream_parser_create. `code` Error code returned by the StreamParser DLL functions, page 50.
Return value	None.
Comment(s)	• –
Code example	–
Version info	Available as of StreamParser DLL V1.0.0.
References	**slsp_stream_parser_connect**, **slsp_stream_parser_set_tcp_timeout**

Name of the function	slsp_stream_parser_set_package_counter
Purpose	Sets the number of Data Packets in stream buffer that trigger a callback handler call.
Function signature	`void slsp_stream_parser_set_package_counter(const slsp_stream_parser* parser, uint32_t new_package_counter, slsp_stream_parser_error_code* code);`
Argument(s)	`parser` Pointer to an StreamParser. Returned by slsp_stream_parser_create. `new_package_counter` Number of Data Packets in stream buffer. Default value: 25. Must be > 0. `code` Error code returned by the StreamParser DLL functions, page 50.
Return value	None.
Comment(s)	• You must call **slsp_stream_parser_set_package_counter** before **slsp_stream_parser_connect**. The `new_package_counter` value can only be set before StreamParser creation. • The `new_package_counter` value cannot be changed at runtime. • With `new_package_counter` = 0 the default value is used. • As soon as the number of Data Packets in stream buffer equals `new_package_counter`: – StreamParser calls the callback handler – StreamParser passes the Data Packets to the callback handler, that is, empties the stream buffer • See also Chapter 3.3 ”Callback Handler Calls by `new_package_counter` and ”Periodic Timer””, page 20.
Code example	–
Version info	Available as of StreamParser DLL V1.1.0.
References	**slsp_stream_parser_connect**, **slsp_stream_parser_set_wait_time_ms**

Name of the function	slsp_stream_parser_set_tcp_timeout	
Purpose	Sets the TCP Connection timeout value.	
Function signature	`void slsp_stream_parser_set_tcp_timeout(const slsp_stream_parser* parser, uint32_t Seconds, slsp_stream_parser_error_code* code);`	
Argument(s)	`parser`	Pointer to an StreamParser. Returned by slsp_stream_parser_create.
	`Seconds`	TCP Connection timeout. value. In s.
	`code`	Error code returned by the StreamParser DLL functions, page 50.
Return value	None.	
Comment(s)	• Default value for the TCP Connection timeout: 10 s. • StreamParser automatically sets `Seconds` = 0 to `Seconds` = 1.	
Code example	–	
Version info	Available as of StreamParser DLL V1.0.0.	
References	**slsp_stream_parser_is_connected**	

Name of the function	slsp_stream_parser_set_wait_time_ms	
Purpose	Changes the interval for the “periodic timer” (see below).	
Function signature	`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(s)	`parser`	Pointer to an StreamParser. Returned by slsp_stream_parser_create.
	`new_wait_time_ms`	The time interval at which the “periodic timer” is called. In ms. Default value: 250 ms. Allowed values: > 0.
	`code`	Error code returned by the StreamParser DLL functions, page 50.
Return value	None.	
Comment(s)	• You must call **slsp_stream_parser_set_wait_time_ms** before **slsp_stream_parser_connect**. The `new_wait_time_ms` value can only be set before StreamParser creation. • The `new_package_counter` value cannot be changed at runtime. • With `new_wait_time_ms` = 0 the default value is used. • The “periodic timer”: – Triggers a callback handler call by StreamParser – no matter how many Data Packets are in stream buffer – StreamParser passes the Data Packets to the callback handler, that is, empties the stream buffer (= callback handler call is made, but stream buffer has `size() == 0`) – Can be used to periodically check if StreamParser is still “alive” – See also Chapter 3.3 ”Callback Handler Calls by `new_package_counter` and “Periodic Timer””, page 20	
Code example	–	
Version info	Available as of StreamParser DLL V1.1.0.	
References	**slsp_stream_parser_connect**, **slsp_stream_parser_set_package_counter**	

6 Function Types

In this Chapter:

- Function type **slsp_stream_callback**

Name of the function type	slsp_stream_callback	
Description	This function type defines: • A Callback Function	
Used by	This function type is used with: • **slsp_stream_parser_create**	
Syntax	`typedef void (*slsp_stream_callback)(slsp_rtc_data_stream* stream, void* context);`	
Argument(s)	`stream`	Data Stream. Pointer.
	`context`	User-defined context. Pointer.
Comment(s)	• **slsp_stream_callback** is called periodically. • After **slsp_stream_callback** has returned the internal Data Stream (which `stream` points to) is cleared. • You must not delete the pointer * `stream`. Only use it within this Callback Function.	
Version info	Available as of StreamParser DLL V1.0.0.	
	Function Types, page 42	

7 Structures struct

In this Chapter:

- `struct` **slsp_rtc_data_package**
- `struct` **slsp_rtc_data_stream**
- `struct` **slsp_rtc_meta_data**
- `struct` **slsp_stream_parser**
- `struct` **slsp_version_info**

Name of the structure	**slsp_rtc_data_package**
Description	This structure defines: • The proxy for a Data Packet.
Used by	This structure is used with: • **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(s)	`slsp_rtc_data_package` Proxy for a Data Packet.
Comment(s)	• **slsp_rtc_data_package** is used as input for the **slsp_data_package[*]** functions.
Version info	Available as of StreamParser DLL V1.0.0.
Structures struct, page 43	

Name of the structure	slsp_rtc_data_stream
Description	This structure defines: • The proxy for a Data Stream.
Used by	This structure is used with: • **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(s)	`slsp_rtc_data_stream` Proxy for a Data Stream.
Comment(s)	• **slsp_rtc_data_stream** is used as input for the **slsp_data_stream[*]** functions.
Version info	Available as of StreamParser DLL V1.0.0.
Structures struct, page 43	

Name of the structure	**slsp_rtc_meta_data**		
Description	This structure defines: • Metadata, page 22 sent by RTC6 Ethernet Board in every Data Packet. • Metadata values are from a random time of the Data Packet recording.		
Used by	This structure is used with: • **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(s)	`uint32_t`	`Period`	• Recording period • `Period` value of the latest set_trigger[*] call
	`uint64_t`	`TimeStamp`	• Timestamp • 64-Bit timestamp made by the RTC6 Ethernet Board upon TCP Packet generation • Corresponds to get_timestamp_long
	`uint32_t`	`SerialNumber`	• Serial number of RTC6 Ethernet Board • Corresponds to get_serial_number.

Name of the structure	slsp_rtc_meta_data		
Argument(s) (cont'd)	uint32_t	OUTVersion	• DSP version number (RTC6ETH.out) • Corresponds to get_hex_version
	uint32_t	RBFVersion	• FPGA version number and options • Corresponds to get_rtc_version
	uint32_t	BiosVersion	• BIOS version number of RTC6 Ethernet Board • Corresponds to get_bios_version
	uint32_t	ListBusy	• RTC6 Ethernet Board status • Corresponds to get_status
	uint32_t	OutputPointer	• Position of output pointer • Corresponds to get_out_pointer
	uint32_t	ExtStartCounter	• Number of triggered External Starts • Corresponds to get_counts
	uint32_t	OverrunCounter	• Number of 10 μs clock cycle overruns • Corresponds to get_overrun
	uint32_t	McBSPIn	• McBSP input value • Corresponds to get_mcbsp
	double	Laptime	• RTC6 Timer value • Corresponds to get_lap_time
	double	Timer	• Saved RTC6 Timer value • Corresponds to get_time
	uint32_t	WaitStatus	• Wait state • Corresponds to get_wait_status
	uint32_t	FlyOffsetX	• x reference value (offset value) for 2D encoder compensation • Corresponds to get_fly_2d_offset(OffsetX)
	uint32_t	FlyOffsetY	• y reference value (offset value) for 2D encoder compensation • Corresponds to get_fly_2d_offset(OffsetY)

Name of the structure	**slsp_rtc_meta_data**		
Argument(s) (cont'd)	`uint16_t`	`ListStartCounter`	• Number of list starts to date • Lower 4 bits corresponds to the value in set_trigger[*] signal 67 Bit #0...Bit #3
	`uint16_t`	`ListStopCounter`	• Number of list stops to date • Lower 4 bits corresponds to the value in set_trigger[*] signal 67 Bit #4...Bit #7
	`uint32_t`	`MasterIP`	• RTC6 DLL IP address • IP address of the PC which is connected by RTC6 DLL, see “Main” User Program. Corresponds to eth_get_card_info Byte 6. Important: Big Endian format!
	`uint32_t`	`HeadStatus`	• XY2-100 status word of iDRIVE scan system • Corresponds to get_head_status(0)
Comment(s)	• The Metadata are sent with every RTC6 Ethernet Board Data Packet. Therefore, its update frequency depends on the settings made for the RTC6 Ethernet Board..		
Version info	Available as of StreamParser DLL V1.0.0.		
Structures struct, page 43			

Name of the structure	**slsp_stream_parser**
Description	This structure defines: • Proxy for the internal StreamParser.
Used by	This structure is used with: • **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(s)	`slsp_stream_parser` Internal StreamParser.
Comment(s)	• –
Version info	Available as of StreamParser DLL V1.0.0.
Structures struct, page 43	

<table>
<tr><th>Name of the structure</th><th colspan="3">slsp_version_info</th></tr>
<tr><td>Description</td><td colspan="3">This structure defines:
• The 3 number blocks of the currently running StreamParser DLL-Version (”Version n.n.n”). See https://semver.org/.</td></tr>
<tr><td>Used by</td><td colspan="3">This structure is used with:
• <b>slsp_get_version_info</b></td></tr>
<tr><td>Syntax</td><td colspan="3"><pre>struct slsp_version_info
{
   uint32_t Major;
   uint32_t Minor;
   uint32_t Patch;
};
typedef struct slsp_version_info slsp_version_info;</pre></td></tr>
<tr><td rowspan="3">Argument(s)</td><td><code>uint32_t</code></td><td><code>Major</code></td><td>Major version of StreamParser DLL.</td></tr>
<tr><td><code>uint32_t</code></td><td><code>Minor</code></td><td>Minor version of StreamParser DLL.</td></tr>
<tr><td><code>uint32_t</code></td><td><code>Patch</code></td><td>Revision version (=Patch version) of StreamParser DLL.</td></tr>
<tr><td>Comment(s)</td><td colspan="3">• –</td></tr>
<tr><td>Version info</td><td colspan="3">Available as of StreamParser DLL V1.0.0.</td></tr>
<tr><td colspan="4">Structures struct, page 43</td></tr>
</table>

8 Enumerated Types enum

In this Chapter:

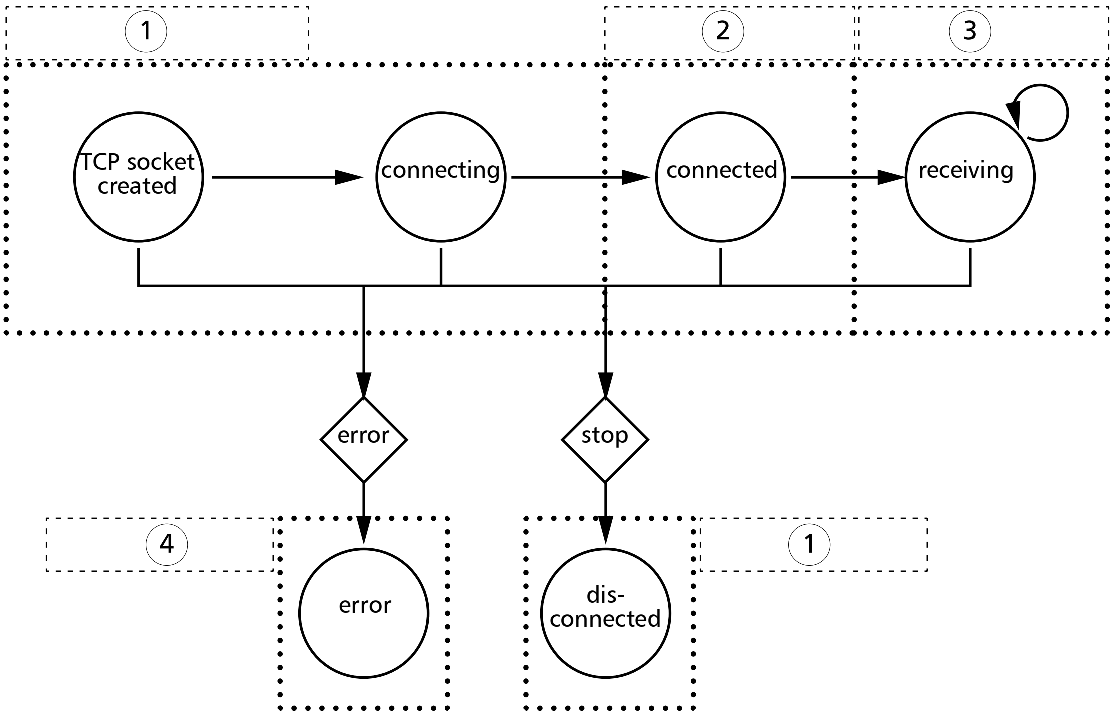
- enum **slsp_stream_parser_error_code**
- enum **slsp_stream_parser_state**

<table>
<tr><th>Name of the enum</th><th colspan="2">slsp_stream_parser_error_code</th></tr>
<tr><td>Description</td><td colspan="2">This enum defines the choices for:
• Error code returned by the StreamParser DLL functions</td></tr>
<tr><td>Used by</td><td colspan="2">This enum is used with:
• All StreamParser DLL-functions except <b>slsp_get_version_info</b></td></tr>
<tr><td>Syntax</td><td colspan="2"><pre>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;</pre></td></tr>
<tr><td rowspan="5">Enumeration constant(s)</td><td>slsp_error_code_OK</td><td>StreamParser DLL-function has been successful.
No error.</td></tr>
<tr><td>slsp_error_code_ESTABLISH_CONNECTION_FAILED</td><td>StreamParser DLL-function failed. A connection to RTC6 Ethernet Board could not be established.</td></tr>
<tr><td>slsp_error_code_RECEIVE_PACKAGE_FAILED</td><td>StreamParser DLL-function failed. An error occurred while trying to receive a Data Packet from RTC6 Ethernet Board. See also page 35.</td></tr>
<tr><td>slsp_error_code_UNKNOWN_ERROR</td><td>StreamParser DLL-function failed. An unspecified error occurred.</td></tr>
<tr><td>slsp_error_code_RTC_VERSION_ERROR</td><td>The RTC6 Ethernet Board does not meet the criteria specified under Hardware prerequisites, page 9 and Software prerequisites, page 9.</td></tr>
<tr><td>Version info</td><td colspan="2">Available as of StreamParser DLL V1.1.0.</td></tr>
<tr><td colspan="3">Enumerated Types enum, page 50</td></tr>
</table>

Name of the enum	**slsp_stream_parser_state**	
Description	This enum defines the choices for: - The state of StreamParser, see Figure 7.	
Used by	This enum is used with: - **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;`	

Enumeration constant(s)	`slsp_STREAM_CONNECTED`	TCP Client is: - Connected to the RTC6 Ethernet Board - Ready to receive the Data Stream. Note that the Callback Function is not called.
	`slsp_STREAM_RECEIVING`	Currently receiving the Data Stream: - The data packets sent from the RTC6 Ethernet Board are incoming - The Callback Function is repeatedly (perpetually) called
	`slsp_STREAM_DISCONNECTED`	Data Stream has been finished properly because: - The RTC6 Ethernet Board has stopped sending data packets - There has been a request to disconnect (**slsp_stream_parser_disconnect**)
	`slsp_STREAM_STOPPED_ON_ERROR`	Data Stream has not been finished properly because: - An error occurred
Version info	Available as of StreamParser DLL V1.0.0.	
Enumerated Types enum, page 50		



Legend

1. StreamParser state `slsp_STREAM_DISCONNECTED`
2. StreamParser state `slsp_STREAM_CONNECTED`
3. StreamParser state `slsp_STREAM_RECEIVING`
4. StreamParser state `slsp_STREAM_STOPPED_ON_ERROR`

7

StreamParser: state machine.

9 Example Code (C++)

- Example Code for RTC6 DLL User Program
- Example Code for StreamParser DLL User Program

9.1 Example Code for RTC6 DLL User Program

```
// Example Code for RTC6 DLL User Program
// activate Data Streaming on RTC6 Ethernet Board side
// not compilable

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

// list definition (set_trigger[*] starts the Data Streaming)
n_set_start_list(Index, 1);

// activate signal recording
n_set_trigger4(Index, 1, 1, 2, 20, 21);

// more code
n_jump_abs(Index, 10, 10);
n_jump_abs(Index, 0,0);
…

// deactivate signal recording (optional, depends on use case)
n_set_trigger4(Index, 0, 1, 2, 20, 21);

// finish list definition and start list execution
n_set_end_of_list(Index);
n_execute_list(CardNo, Index);
```

9.2 Example Code for StreamParser DLL User Program

A compilable example code for a StreamParser DLL User Program is available in StreamParser Software Package under:

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

10 Change Index

The following are changes in this manual due to the technical evolution of the product as well as significant editorial changes.

Document revision 0.0.9 en-US

Where	What
Global	Document Revision • 0.0.9 en-US applies to StreamParser DLL • V0.5.0.RC3
Global	Preliminary version.

Document revision to 1.0.0 en-US from 0.0.9 en-US

Where	What
Global	Document Revision • 1.0.0 en-US applies to StreamParser DLL • V1.0.0
Chapter 9.2 ”Example Code for StreamParser DLL User Program”, page 53	Editorial enhancement.

Document revision to 1.1.0 en-US from 1.0.0 en-US

Where	What
Global	Document Revision • 1.1.0 en-US applies to StreamParser DLL • V1.1.0
Global	Software change. Designation "Extended Scan Head Status". Renamed from "Low Bandwidth Return Channel Multiplexing".
Chapter 2.4 "Software Package Content", page 11	Software change.
Chapter 3 "Software Development with StreamParser DLL", page 12	Editorial enhancement. Figure 1, page 14. Figure 2, page 15. Figure 3, page 16. Figure 4, page 17.
Chapter 3.3 "Callback Handler Calls by `new_package_counter` and "Periodic Timer"", page 20	Software change.
slsp_stream_parser_get_async_error, page 38	Software change. Renamed from **slsp_stream_parser_get_asnc_error**.
slsp_stream_parser_set_package_counter, page 39	Software change. New function.
slsp_stream_parser_set_wait_time_ms, page 41	Software change. New function.
slsp_stream_parser_error_code, page 50	Software change. New enumeration constant `slsp_error_code_RTC_VERSION_ERROR`.

SCANLAB

innovators for industry

Notes