TECHNICAL INFORMATION

API Description

LiDAR localization

SICK
Sensor Intelligence.

Described product

API Description

Manufacturer

SICK AG
Erwin-Sick-Str. 1
79183 Waldkirch
Germany

Legal information

Original document

This document is an original document of SICK AG.

Contents

1 About this document

1.1 Function of this document

This document describes the API (**A**pplication **P**rogramming **I**nterface) to communicate with LiDAR-LOC. The interfaces described are:

- CoLa-A (**Co**mmand **La**nguage **A**SCII) - deprecated (SIMs only)
- CoLa 2 (**Co**mmand **La**nguage **2**)
- REST (**RE**presentational **S**tate **T**ransfer)
- ROS (**R**obot **O**perating **S**ystem)

The full description of the ROS driver can be found here:

- https://github.com/SICKAG/sick_lidar_localization

The full operating instructions stated below are explicitly required for the use of LIDAR-LOC:

- Operating instructions (8025192)
- Hardware Integration (8025193)

NOTE

Some commands may change during SICK development processes. Please always use the latest version of the “API Description” document.

2 User interfaces

Overview

You can control, set, and call all localization functions with different user interfaces.

There are the following three interface types:

- GUI: Graphical User Interface
- Command interface
- Streaming interface

Status requests, situational information or configuration changes are exchanged between the localization controller and the vehicle controller via the command interface. After a restart of the localization controller, configuration changes are reset. Permanent configuration changes are saved in the configuration file.

With the streaming interface, the localization controller receives and sends data continuously to the external vehicle controller.

NOTE
The usage of the generic C++ driver in the vehicle controller can replace the command API CoLa-A, CoLa2 and REST as well as the streaming API (UDP). With this driver the LiDAR-LOC API is fully integrated.

Interfaces and ports

Table 1: Overview of the interfaces

Usage	Interface	Interface type	Source port	Destination port	Description
SOPASair web server	HTTP	GUI	80[1)]	80[1)]	Control and setting of all localization functions via the web-interface.
Configuration in operation	REST (HTTP)	Command	80[1)]	80[1)]	Control of localization functions via REST API.
Configuration in operation	CoLa-A Deprecated (TCP)	Command	2111 / 2112	2111 / 2112	Control of localization functions via ASCII telegrams (only SICK controllers, SIMs). CoLa is a SICK proprietary format based on TCP/IP.
Configuration in operation	CoLa2 (TCP)	Command	Selected by CoLa2 client (e.g. vehicle controller)	2122	Control of localization functions via binary CoLa2 telegrams. CoLa is a SICK proprietary format based on TCP/IP.
Generic (C++)	UDP, REST	Command and Streaming	See corresponding protocol	See corresponding protocol	The usage of the generic C++ driver in the vehicle controller can replace the command API CoLa-A, CoLa2 and REST as well as the streaming API (UDP). With this driver the LiDAR-LOC API is fully integrated.
ROS	TCP, UDP, REST	Command and Streaming	-	-	When using ROS in the vehicle controller, the command API CoLa-A, CoLa2 and REST as well as the streaming data (UDP) can be replaced with the ROS driver.
Data input	UDP	Streaming	Selected by UDP client (e.g. LiDAR-LOC)	5009[2)]	Streaming data to receive partly pre-processed sensor data of the external vehicle controller from 2D LiDAR sensor, wheel odometry, line and code sensors
Data output	UDP	Streaming	5008[2)]	5010[2)]	Streaming data to output localization results.
Data transfer	FTP	On request	2300[2)]	2300[2)]	Access to public directory to retrieve data, e.g. support data recording.

Usage	Interface	Interface type	Source port	Destination port	Description
Measurement data LiDAR safety sensor	UDP	Streaming	Randomly selected	Configurable[3)]	Continuous measurement data output of the sensor to the localization controller.
Configuration of LiDAR safety sensor	CoLa2 (TCP)	Configuration	Selected by CoLa2 client (e.g. LiDAR-LOC)	2122	Configuration of the measurement data output of the safety LiDAR sensor by the localization controller
Measurement data of LiDAR sensor	TCP, CoLa-A	Configuration and Streaming	Randomly selected	2111/2112	Continuous measurement data output of the sensor to the localization controller. Configuration of the measurement data output of the LiDAR sensor by the localization controller
Measurement data picoScan150	UDP	Streaming	Randomly selected	Configurable, default 2115	Continuous measurement data output of the sensor to the localization controller. Configuration of the measurement data output of the LiDAR sensor by the localization controller or by user. MSGPCK and Compact format.

1) Configurable for IPCs: Changing the web server port
2) Configured in `lidarloc_config.yml` or in SOPASair Web GUI setup in `communication`.
3) Configured in `lidarloc_config.yml` or in SOPASair Web GUI setup in `sensors -> lidars -> interface -> receiverPort`.
   - `Automatic`: A random port 51000 to 51050 is selected.
   - `Manual`: A specified port is selected, for example 56661.

   Can also be configured in the Safety Designer (Configuration -> Data output -> UDP port), e.g. 6060. The same value must then also be set in LiDAR-LOC.

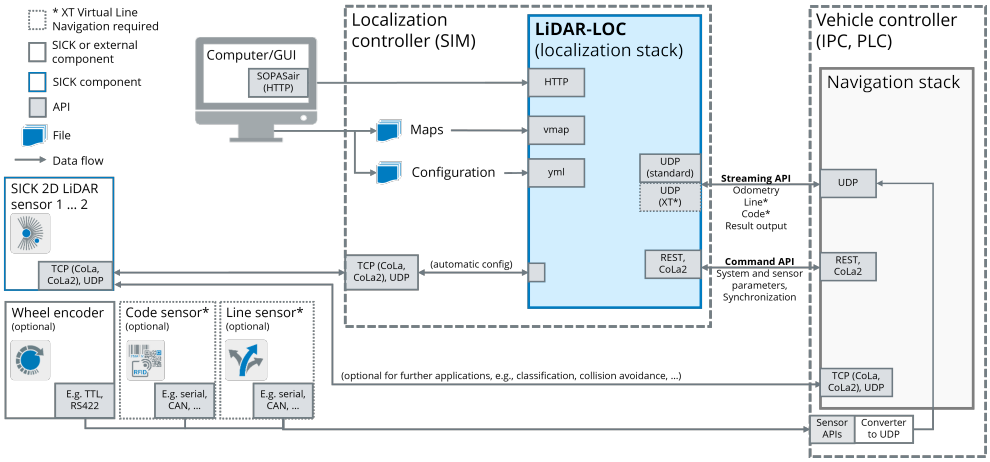

Figure 1: Overview of the interfaces using SICK controller (SIM)

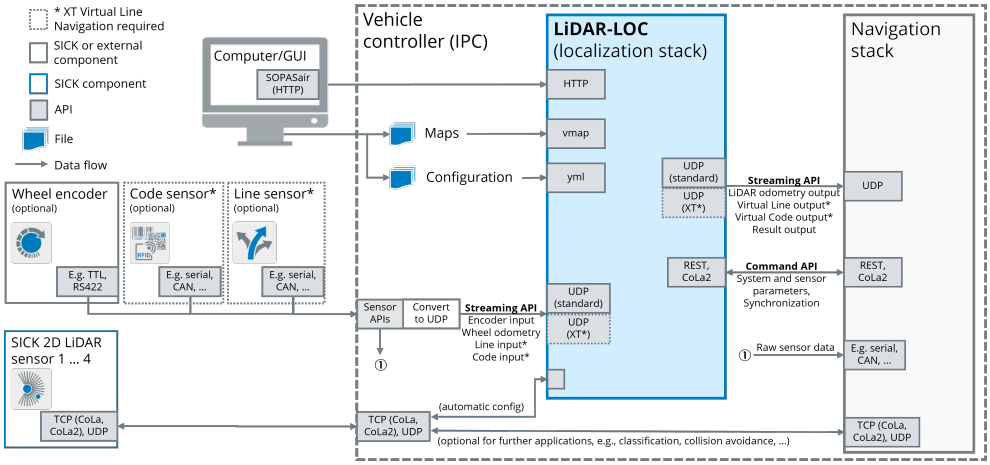

Figure 2: Overview of the interfaces using industrial PC (IPC)

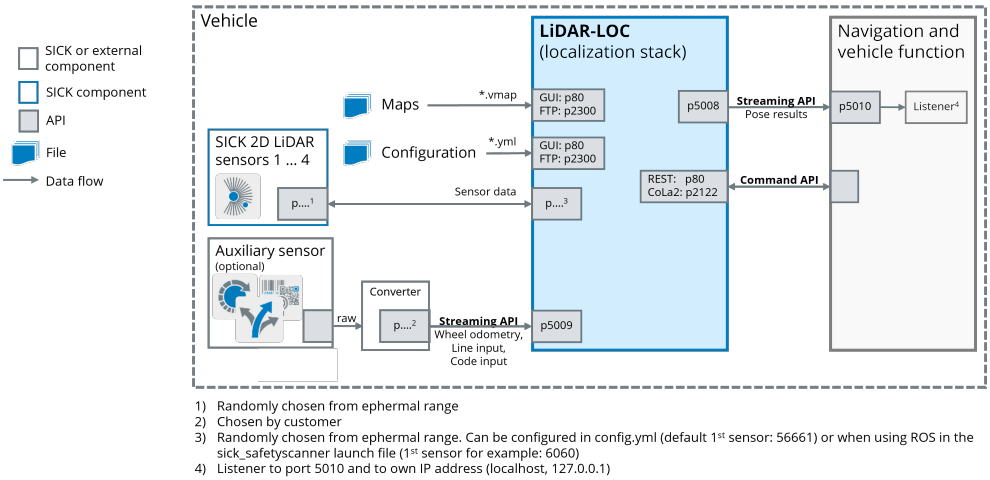

Figure 3: Overview of the ports with auxiliary sensors

Complementary information

- You can find the required generic C++ and ROS driver as well as the documentation on GitHub:
  - https://github.com/SICKAG/sick_lidar_localization
- LiDAR-LOC Operating Instructions, 8027121

3 Command API

Overview

Status requests, situational information or configuration changes are exchanged between the localization controller and the vehicle controller via the command interface. After a restart of the localization controller, configuration changes are reset. Permanent configuration changes are saved in the configuration file.

The command API provides the following interfaces:

- REST
- CoLa-A Deprecated
- CoLa2
- ROS

Example start sequence

The simplified start sequence (figure 4) can be used if the localization system was previously successfully configured via SOPASair or API.

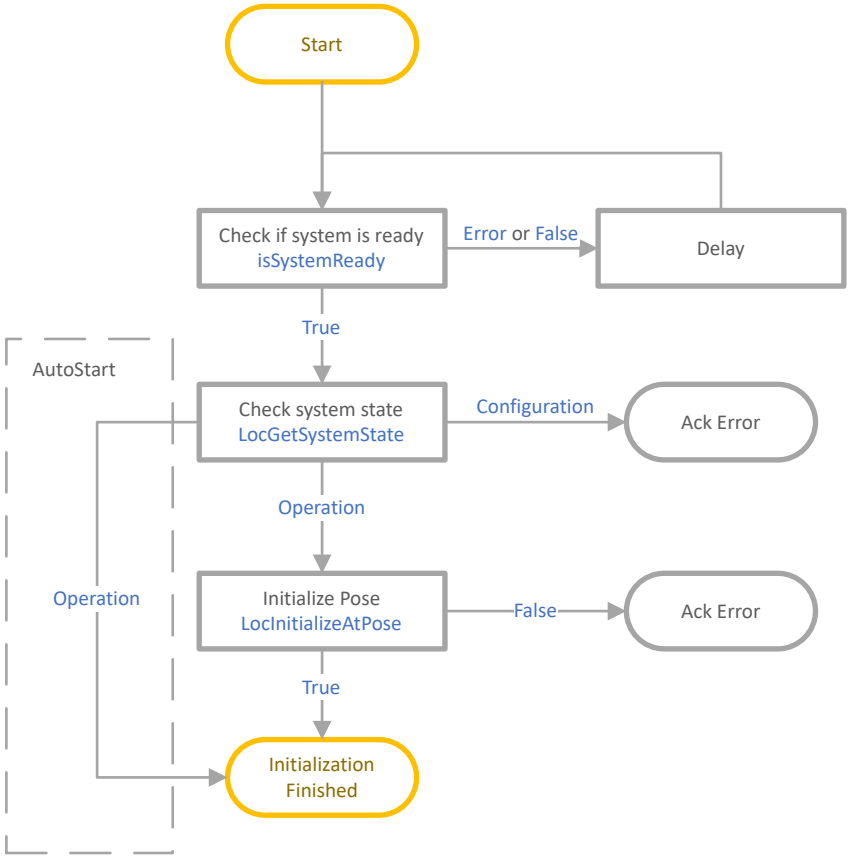

Figure 4: Simplified start sequence (commands and results are written in blue.)

3.1 System

3.1.1 Command: GetSoftwareVersion

Overview

Get the current software version of the localization controller's application software.

Request

Table 2: Request command

Type	Command
Getter	GetSoftwareVersion

Table 3: Request data

Data field	Type	Description	Range/Value
-	-	-	-

Example

- Read the current software version.

Table 4: Request example

Interface	Type	Request example
CoLa-A Deprecated	sMN	`sMN GetSoftwareVersion`
CoLa2	MN	`MNGetSoftwareVersion`
REST	POST	`http://192.168.0.1/api/GetSoftwareVersion`
Generic (C++)	gen_srv	`./build/gen_service_call LocGetSoftwareVersion POST "{}" -d=2`
ROS 1	srv	`rosservice call LocGetSoftwareVersion "{}"`
ROS 2		`ros2 service call LocGetSoftwareVersion sick_lidar_localization/srv/LocGetSoftwareVersion Srv "{}"`

Response

Table 5: Response data

Data field	Type	Description	Range/Value
version	String	The software version.	Max length = 255

Example

- The current software version is 2.0.0.14R.

Table 6: Response example

Interface	Response example
CoLa-A Deprecated	**Hexadecimal**[1)] `sAN GetSoftwareVersion 9 2.0.0.14R`

CoLa2	**ISO 8859-15**[1)]
	`ANGetSoftwareVersion 0x00092.0.0.14R`
	Hexadecimal[1)]
	`414E 4765 7453 6F66 7477 6172 6556 6572 7369 6F6E 2000 0932 2E30 2E30 2E31 3452`

Interface	Response example

REST	<pre>{
	  "header": {
	  "status": 0,
	  "message": "Ok"
	  },
	  "data": {
	  "version": "2.0.0.14R"
	  }
	}</pre>

Generic (C++)	<pre>response: "{"header":
	{"status":0,"message":"Ok"},"data":{"success":true}}"</pre>

ROS 1	<pre>success=1</pre>
ROS 2	

1) The string has a prefix that specifies the length of the string. In this case the character length is 0x9, 9_{dec}.

3.1.2 Command: IsSystemReady

Overview

Checks if the localization controller is booted and ready to process commands.

Request

Table 7: Request command

Type	Command
Getter	IsSystemReady

Table 8: Request data

Data field	Type	Description	Range/Value
-	-	-	-

Example

- Check if the system is ready.

Table 9: Request example

Interface	Type	Request example
CoLa-A Deprecated	sMN	`sMN IsSystemReady`
CoLa2	MN	`MNIsSystemReady`
REST	POST	`http://192.168.0.1/api/IsSystemReady`
Generic (C++)	gen_srv	`./build/gen_service_call LocIsSystemReady POST "{}" -d=2`
ROS 1	srv	`rosservice call LocIsSystemReady "{}"`
ROS 2		`ros2 service call LocIsSystemReady sick_lidar_localization/srv/LocIsSystemReadySrv "{}"`

Response

Table 10: Response data

Data field	Type	Description	Range/Value
state	Bool_1	The localization controller is ready for communication	No answer or error: Localization controller is booting or LiDAR-LOC is loading 1 true: Localization controller is ready

Example 1:

- The system is ready.

Table 11: Response example

Interface	Response example
CoLa-A Deprecated	`sAN IsSystemReady 1`

CoLa2	**ISO 8859-15**
	`ANIsSystemReady 1`
	Hexadecimal
	`414E 4973 5379 7374 656D 5265 6164 7920 01`

Interface	Response example

REST	<pre>{
	"header": {
	"status": 0,
	"message": "Ok"
	},
	"data": {
	"success": true
	}
	}</pre>

Generic (C++)	<pre>response: "{"header":
	{"status":0,"message":"Ok"},"data":{"success":true}}"</pre>

ROS 1	<pre>success=1</pre>
ROS 2	

Example 2:

- The system is not yet booted and ready.

Table 12: Response example

Interface	Response example
CoLa-A Deprecated	<pre>no answer or error</pre>
CoLa2	<pre>no answer or error</pre>

REST	<pre>{
	"header": {
	"status": 3,
	"message": "Internal Server Error"
	}
	}</pre>

Generic (C++)	<pre>response: "{"header":{"status":3,"message":"Internal
	Server Error"}}"</pre>

ROS 1	<pre>no answer or error</pre>
ROS 2	

3.1.3 Command: LoadPersistentConfig

Overview

Reload the persistent LiDAR-LOC configuration file and discard the current changes.

Prerequisites

- The localization software is in system state **Configuration**.

Request

Table 13: Request command

Type	Command
Setter	LoadPersistentConfig

Table 14: Request data

Data field	Type	Description	Range/Value
-	-	-	-

Example

- Load the persistent config and discard the current changes.

Table 15: Request example

Interface	Type	Request example
CoLa-A Deprecated	sMN	`sMN LoadPersistentConfig`
CoLa2	MN	`MNLoadPersistentConfig`
REST	POST	`http://192.168.0.1/api/LoadPersistentConfig`
Generic (C++)	gen_srv	`./build/gen_service_call LocLoadPersistentConfig POST "{}" -d=2`
ROS 1	srv	`rosservice call LocLoadPersistentConfig "{}"`
ROS 2		`ros2 service call LocLoadPersistentConfig sick_lidar_localization/srv/LocLoadPersistentConfigSrv "{}"`

Response

Table 16: Response data

Data field	Type	Description	Range/Value
set result	Bool_1	The command was valid and has been executed.	0 failed 1 success

Example

- The persistent parameters have been loaded successfully and the temporary changes have been discarded.

Table 17: Response example

Interface	Response example
CoLa-A Deprecated	`sAN LoadPersistentConfig 1 0`

CoLa2	**ISO 8859-15**
	`ANLoadPersistentConfig 1 0`
	Hexadecimal
	`414E 4C6F 6164 5065 7273 6973 7465 6E74 436F 6E66 6967 2001 2000`

Interface	Response example

REST	<pre>{
	    "header": {
	        "status": 0,
	        "message": "Ok"
	    },
	    "data": {
	        "success": true
	        "response": ""
	    }
	}</pre>

Generic (C++)	<pre>response: "{"header":
	{"status":0,"message":"Ok"},"data":{"success":true,
	"response": ""}}"</pre>

ROS 1	<pre>success=1</pre>
ROS 2	<pre>response=""</pre>

3.1.4 Command: LocGetSystemState

Overview

Read the current state of the localization state machine.

Request

Table 18: Request command

Type	Command
Getter	LocGetSystemState

Table 19: Request data

Data field	Type	Description	Range/Value
-	-	-	-

Example

- Read the state of the system.

Table 20: Request example

Interface	Type	Request example
CoLa-A Deprecated	sMN	`sMN LocGetSystemState`
CoLa2	MN	`MNLocGetSystemState`
REST	POST	`http://192.168.0.1/api/LocGetSystemState`
Generic (C++)	gen_srv	`./build/gen_service_call LocGetSystemState POST "{}" -d=2`
ROS 1	srv	`rosservice call LocGetSystemState "{}"`
ROS 2		`ros2 service call LocGetSystemState sick_lidar_localization/srv/LocGetSystemStateSrv "{}"`

Response

Table 21: Response data

Data field	Type	Description	Range/Value
systemState	Enum	Localization state.	• Configuration • Operation

Example

- LiDAR-LOC is in "Operation" mode.

Table 22: Response example

Interface	Response example
CoLa-A Deprecated	**Hexadecimal**[1)] `sAN LocGetSystemState 9 Operation`

CoLa2	**ISO 8859-15**[1)]
	`ANLocGetSystemState 0x0009Operation`
	Hexadecimal[1)]
	`414E 4C6F 6347 6574 5379 7374 656D 5374 6174 6520 00094F70 6572 6174 696F 6E`

Interface	Response example

REST	<pre>{
	  "header": {
	  "status": 0,
	  "message": "Ok"
	  },
	  "data": {
	  "systemState": Operation
	  }
	}</pre>

Generic (C++)	<pre>response: {"header":
	{"status":0,"message":"Ok"},"data{"systemState":"Operation"}}</pre>

ROS 1	<pre>systemstate: "Operation"</pre>
ROS 2	

1) The string has a prefix that specifies the length of the string. In this case the character length is 0x9, 9_{dec}.

3.1.5 Command: LocStartLocalizing

Overview

Start the localization.

NOTICE
This command is deprecated. Use **LocStart** instead.

Request

Table 23: Request command

Type	Command
Setter	LocStartLocalizing

Table 24: Request data

Data field	Type	Description	Range/Value
-	-	-	-

Example

- Start the localization.

Table 25: Request example

Interface	Type	Request example
CoLa-A Deprecated	sMN	`sMN LocStartLocalizing`
REST	POST	`http://192.168.0.1/api/LocStartLocalizing`
Generic (C++)	gen_srv	`./build/gen_service_call LocStartLocalizing POST "{}" -d=2`
ROS 1	srv	`rosservice call LocStartLocalizing "{}"`
ROS 2		`ros2 service call LocStartLocalizing sick_lidar_localization/srv/LocStartLocalizingSrv "{}"`

Response

Table 26: Response data

Data field	Type	Description	Range/Value
set result	Bool_1	The command was valid and has been executed.	0 failed 1 success

Example

- The system started localizing.

Table 27: Response example

Interface	Response example
CoLa-A Deprecated	`sAN LocStartLocalizing 1`
REST	`{ "header": { "status": 0, "message": "Ok" }, "data": { "success": true } }`

Interface	Response example
Generic (C++)	`response: "{"header": {"status":0,"message":"Ok"},"data":{"success":true}}"`
ROS 1	`success=1`
ROS 2	

3.1.6 Command: LocStart

Overview

Set the system to state **Operation**.

Request

Table 28: Request command

Type	Command
Setter	LocStart

Table 29: Request data

Data field	Type	Description	Range/Value
-	-	-	-

Example

- Set the system to state **operation**.

Table 30: Request example

Interface	Type	Request example
CoLa-A Deprecated	sMN	`sMN LocStart`
CoLa2	MN	`MNLocStart`
REST	POST	`http://192.168.0.1/api/LocStart`
Generic (C++)	gen_srv	`./build/gen_service_call LocStart POST "{}" -d=2`
ROS 1	srv	`rosservice call LocStart "{}"`
ROS 2		`ros2 service call LocStart sick_lidar_localization/srv/LocStartSrv "{}"`

Response

Table 31: Response data

Data field	Type	Description	Range/Value
set result	Bool_1	The command was valid and has been executed.	0 failed 1 success

Example

- The system is in operation.

Table 32: Response example

Interface	Response example
CoLa-A Deprecated	`sAN LocStart 1`

CoLa2	**ISO 8859-15**
	`ANLocStart 0x01`
	Hexadecimal
	`414E 4C6F 6353 7461 7274 2001`

REST	`{ "header": { "status": 0, "message": "Ok" }, "data": { "success": true } }`

Interface	Response example
Generic (C++)	`response: "{"header": {"status":0,"message":"Ok"},"data":{"success":true}}"`
ROS 1	`success=1`
ROS 2	

3.1.7 Command: LocStop

Overview

Stop the localization and return to **Configuration** state.

Request

Table 33: Request command

Type	Command
Setter	LocStop

Table 34: Request data

Data field	Type	Description	Range/Value
-	-	-	-

Example

- Stop the localization.

Table 35: Request example

Interface	Type	Request example
CoLa-A Deprecated	sMN	`sMN LocStop`
CoLa2	MN	`MNLocStop`
REST	POST	`http://192.168.0.1/api/LocStop`
Generic (C++)	gen_srv	`./build/gen_service_call LocStop POST "{}" -d=2`
ROS 1	srv	`rosservice call LocStop "{}"`
ROS 2		`ros2 service call LocStop sick_lidar_localization/srv/LocStopSrv "{}"`

Response

Table 36: Response data

Data field	Type	Description	Range/Value
set result	Bool_1	The command was valid and has been executed.	0 failed 1 success

Example

- The localization has been stopped.

Table 37: Response example

Interface	Response example
CoLa-A Deprecated	`sAN LocStop 1`

CoLa2	**ISO 8859-15**
	`ANLocStop 0x01`
	Hexadecimal
	`414E 4C6F 6353 746F 7020 01`

REST	<pre>{
	  "header": {
	  "status": 0,
	  "message": "Ok"
	  },
	  "data": {
	  "success": true
	  }
	}</pre>

Interface	Response example
Generic (C++)	`response: "{"header": {"status":0,"message":"Ok"},"data":{"success":true}}"`
ROS 1	`success=1`
ROS 2	

3.1.8 Command: LocGetLocalizationStatus

Overview

Read the current localization status parameter.

NOTE
To receive continous localization status data, use the continous UDP streaming message Localization result - MsgType: 5.

Request

Table 38: Request command

Type	Command
Getter	LocGetLocalizationStatus

Table 39: Request data

Data field	Type	Description	Range/Value
-	-	-	-

Example

- Read the localization status parameter.

Table 40: Request example

Interface	Type	Request example
CoLa-A Deprecated	sMN	`sMN LocGetLocalizationStatus`
CoLa2	MN	`MNLocGetLocalizationStatus`
REST	POST	`http://192.168.0.1/api/LocGetLocalizationStatus`
Generic (C++)	gen_srv	`./build/gen_service_call LocGetLocalizationStatus POST "{}" -d=2`
ROS 1	srv	`rosservice call LocGetLocalizationStatus "{}"`
ROS 2		`ros2 service call LocGetLocalizationStatus sick_lidar_localization/srv/LocGetLocalizationStatusSrv "{}"`

Response

Table 41: Response data

Data field	Type	Description	Range/Value
locStatus	UInt_8	The current localization status.	10: OK 20: Warning 30: Not Localized 40: System Error 100: Undefined 1)
details	String	Further remarks on current localization status.	Max length = 10000000

1) Return value 100 when system state is in **Configuration**.

Example: Localization Status OK

- LiDAR-LOC is in localization status is 0xA, 10_{dec}, \n with the meaning **OK**.

Table 42: Response example: Ok

Interface	Response example
CoLa-A Deprecated	Hexadecimal[1)] `sAN LocGetLocalizationStatus A 0`

Interface	Response example

CoLa2	**ISO 8859-15**[1)]
	`ANLocGetLocalizationStatus 0x0A 0x0000`
	Hexadecimal[1)]
	`414E 4C6F 6347 6574 4C6F 6361 6C69 7A61 7469 6F6E 5374 6174 7573 200A 2000 00`

REST	`{ "header": { "status": 0, "message": "Ok" }, "data": { "locStatus": 10, "details": "" } }`
Generic (C++)	`response: {"header": {"status":0,"message":"Ok"},"data{"locStatus":10,"details": ""}}`
ROS 1 ROS 2	`locStatus: 10` `details: ""`

1) The second hex digit specifies the length of the answer string. In this case the character length is 0x0000, 0_{dec} since there is no further detail description.

Example: Localization Status Warning

- LiDAR-LOC is in localization status is 0x14, 20_{dec}, \n with the meaning **Warning**.

Table 43: Response example: Warning

Interface	Response example

CoLa-A
Deprecated
	Hexadecimal[1)]
	`sAN LocGetLocalizationStatus 14 37`
	`ReferenceScanLocalizer: Covariance over warning level;`

CoLa2	**ISO 8859-15**[1)]
	`ANLocGetLocalizationStatus 0x14 0x0037`
	`ReferenceScanLocalizer: Covariance over warning level;`
	Hexadecimal[1)]
	`414E 4C6F 6347 6574 4C6F 6361 6C69 7A61 7469 6F6E 5374 6174 7573 2014 2000 3752 6566 6572 656E 6365 5363 616E 4C6F 6361 6C69 7A65 723A 2043 6F76 6172 6961 6E63 6520 6F76 6572 2077 6172 6E69 6E67 206C 6576 656C 3B20`

REST	`{ "header": { "status": 0, "message": "Ok" }, "data": { "locStatus": 20, "details": "ReferenceScanLocalizer: Covariance over warning level; " } }`

Interface	Response example
Generic (C++)	`response: {"header": {"status":0,"message":"Ok"},"data{"locStatus":20,"details": "ReferenceScanLocalizer: Covariance over warning level; "}}`
ROS 1	`locStatus: 20 details: "ReferenceScanLocalizer: Covariance over warning level; "`
ROS 2	

1) The second hex digit specifies the length of the answer string. In this case the character length is 0x37, 55_{dec} including the semicolon and ending space.

3.2 Mapping

3.2.1 Command: LocSetMappingActive

Overview

Enable or disable the recording of scan data for map creation.

Request

Table 44: Request command

Type	Command
Setter	LocSetMappingActive

Table 45: Request data

Data field	Type	Description	Range/Value
state	Bool_1	True to activate, false to deactivate the recording scan data for map creation.	0 false 1 true

Example

- Set the recording of scan data for map creation active.

Table 46: Request example

Interface	Type	Request example
CoLa-A Deprecated	sMN	`sMN LocSetMappingActive 1`
CoLa2	MN	`MNLocSetMappingActive 1`

REST	POST	`http://192.168.0.1/api/LocSetMappingActive`
		`{`
		`"data":`
		`{`
		`"active": true`
		`}`
		`}`

Generic (C++)	gen_srv	`./build/gen_service_call LocSetMappingActive POST "{\"data\": {\"active\":true}}" -d=2`
ROS 1	srv	`rosservice call LocSetMappingActive "{active: true}"`
ROS 2		`ros2 service call LocSetMappingActive sick_lidar_localization/srv/LocSetMappingActiveSrv "{active: true}"`

Response

Table 47: Response data

Data field	Type	Description	Range/Value
set result	Bool_1	The command was valid and has been executed.	0 failed 1 success

Example

- The recording of scan data for map creation has been started.

Table 48: Response example

Interface	Response example
CoLa-A Deprecated	`sAN LocSetMappingActive 1`

Interface	Response example

CoLa2	**ISO 8859-15**
	`ANLocSetMappingActive 0x01`
	Hexadecimal
	`414E 4C6F 6353 6574 4D61 7070 696E 6741 6374 6976 6520 01`

REST	`{ "header": { "status": 0, "message": "Ok" }, "data": { "success": true } }`
Generic (C++)	`response: "{"header": {"status":0,"message":"Ok"},"data":{"success":true}}"`
ROS 1	`success=1`
ROS 2	

3.2.2 Command: LocSetMap

Overview

Load and activate a map.

The localization status is set to **Not localized**. If the parameter dataProcessing -> localizer -> stopWhenDelocalized is set to false, the initialization can be triggered. If the contour is sufficient and the coordinate system is the same, the localization status changes back to **Ok**.

Prerequisites

- A reference map has been created with the SICK Map Engineering Tool (SMET) and transferred on the localization controller.

Request

Table 49: Request command

Type	Command
Setter	LocSetMap

Table 50: Request data

Data field	Type	Description	Range/Value
mapPath	String	The file name of the map that should be loaded.	Max length 255

Example

Load a map with the following parameters:

- Map file name: "mytest.vmap"

Table 51: Request example

Interface	Type	Request example

CoLa-A
Deprecated
	sMN	Hexadecimal length[1)]
		`sMN LocSetMap B mytest.vmap`
		Decimal length[1)2)]
		`sMN LocSetMap +11 mytest.vmap`

CoLa2	MN	ISO 8859-15[1)]
		`MNLocSetMap 0x000Bmytest.vmap`
		Hexadecimal
		`4D4E 4C6F 6353 6574 4D61 7020 000B 6D79 7465 7374 2E76 6D61 70`

REST	POST	`http://192.168.0.1/api/LocSetMap` `{ "data": { "mapPath": "mytest.vmap" } }`
Generic (C++)	gen_srv	`./build/gen_service_call LocSetMap POST "{\"data\": {\"mapPath\": \"mytest.vmap\"}}" -d=2`
ROS 1	srv	`rosservice call /LocSetMap "{mappath: \"mytest.vmap\"}"`
ROS 2		`ros2 service call /LocSetMap sick_lidar_localization/srv/LocSetMapSrv "{mappath: \"mytest.vmap\"}"`

1) The string has a prefix that specifies the length of the string. In this case the character length is 0xB, 11_{dec}.
2) To use the decimal notation, a "+" must be prefixed.

Response

Table 52: Response data

Data field	Type	Description	Range/Value
set result	Bool_1	The command was valid and has been executed.	0 failed 1 success

Example

- The map was available and has been activated.

Table 53: Response example

Interface	Response example
CoLa-A Deprecated	`sAN LocSetMap 1`

CoLa2	**ISO 8859-15**
	`ANLocSetMap 1`
	Hexadecimal
	`414E 4C6F 6353 6574 4D61 7020 01`

REST	`{`
	`"header": {`
	`"status": 0,`
	`"message": "Ok"`
	`},`
	`"data": {`
	`"success": true`
	`}`
	`}`

Generic (C++)	`response: "{"header": {"status":0,"message":"Ok"},"data":{"success":true}}"`
ROS 1	`success=1`
ROS 2	

Complementary information

Corresponding relevant configuration parameters of the configuration file:

```
map:
  file: test.vmap
```

3.2.3 Command: LocSwitchMap

Overview

If the pose is close enough to a transition point between the current and the passed map, switches **LocSwitchMap** to the passed map. The transition must be defined in SMET in a *.vmaptr file. The localization status is not changed.

Use **LocLoadMapToCache** to accelerate the loading time.

Prerequisites

- A reference map has been created with the SICK Map Engineering Tool (SMET) and transferred on the localization controller.
- A transition file has been created with the SICK Map Engineering Tool (SMET) and transferred on the localization controller.

Request

Table 54: Request command

Type	Command
Setter	LocSwitchMap

Table 55: Request data

Data field	Type	Description	Range/Value
subMapName	String	The file name of the map that should be loaded.	Max length 255

Example

Load a map with the following parameters:

- Map file name: "mytest.vmap"

Table 56: Request example

Interface	Type	Request example

CoLa-A
Deprecated
	sMN	**Hexadecimal length**[1)]
		`sMN LocSwitchMap B mytest.vmap`
		Decimal length[1)2)]
		`sMN LocSwitchMap +11 mytest.vmap`

CoLa2	MN	**ISO 8859-15**[1)]
		`MNLocSwitchMap 0x000Bmytest.vmap`
		Hexadecimal[1)]
		`4D4E 4C6F 6353 7769 7463 684D 6170 2000 0B6D 7974 6573 742E 766D 6170`

REST	POST	`http://192.168.0.1/api/LocSwitchMap`
		`{`
		`"data": {`
		`"subMapName": "mytest.vmap"`
		`}`
		`}`

Generic (C++)	gen_srv	`./build/gen_service_call LocSwitchMap POST "{\"data\": {\"subMapName\":\"mytest.vmap\"}}" -d=2`

Interface	Type	Request example
ROS 1	srv	`rosservice call /LocSwitchMap {submapname: \"mytest.vmap\"}"`
ROS 2		`ros2 service call /LocSwitchMap sick_lidar_localization/srv/LocSwitchMapSrv "{submapname: \"mytest.vmap\"}"`

1) The string has a prefix that specifies the length of the string. In this case the character length is 0xB, 11_{dec}.
2) To use the decimal notation, a "+" must be prefixed.

Response

Table 57: Response data

Data field	Type	Description	Range/Value
set result	Bool_1	The command was valid and has been executed.	0 failed 1 success

Example

- The map was available and has been activated.

Table 58: Response example

Interface	Response example
CoLa-A Deprecated	`sAN LocSwitchMap 1`

CoLa2	**ISO 8859-15**
	`ANLocSwitchMap 1`
	Hexadecimal
	`414E 4C6F 6353 7769 7463 684D 6170 2001`

REST	`{ "header": { "status": 0, "message": "Ok" }, "data": { "success": true } }`
Generic (C++)	`response: "{"header": {"status":0,"message":"Ok"},"data":{"success":true}}"`
ROS 1	`success=1`
ROS 2	

3.2.4 Command: LocLoadMapToCache

Overview

Loads a map to the RAM of the localization controller if not already loaded. It does not set the map for localization. However, if you call LocSetMap or LocSwitchMap after this call the time for changing the map is reduced by the time it takes to load the map from the disk.

Prerequisites

- A reference map has been created with the SICK Map Engineering Tool (SMET) and transferred on the localization controller.

Request

Table 59: Request command

Type	Command
Setter	LocLoadMapToCache

Table 60: Request data

Data field	Type	Description	Range/Value
mapPath	String	The file name of the map that should be loaded.	Max length 255

Example

Load a map with the following parameters:

- Map file name: "mytest.vmap"

Table 61: Request example

Interface	Type	Request example

CoLa-A
Deprecated
	sMN	**Hexadecimal length**[1]
		`sMN LocLoadMapToCache B mytest.vmap`
		Decimal length[1)2)]
		`sMN LocLoadMapToCache +11 mytest.vmap`

CoLa2	MN	**ISO 8859-15**[1]
		`MNLocLoadMapToCache 0x000Bmytest.vmap`
		Hexadecimal[1]
		`4D4E 4C6F 634C 6F61 644D 6170 546F 4361 6368 6520 000B 6D79 7465 7374 2E76 6D61 70`

REST	POST	`http://192.168.0.1/api/LocLoadMapToCache` `{ "data": { "mapPath": "mytest.vmap" } }`
Generic (C++)	gen_srv	`./build/gen_service_call LocLoadMapToCache POST "{\"data\": {\"mappath\": \"mytest.vmap\"}}" -d=2`
ROS 1	srv	`rosservice call LocLoadMapToCache "{mappath: \"mytest.vmap\"}"`
ROS 2		`ros2 service call LocLoadMapToCacheMap sick_lidar_localization/srv/LocLoadMapToCacheSrv "{mappath: \"mytest.vmap\"}"`

1) The string has a prefix that specifies the length of the string. In this case the character length is 0xB, 11_{dec}.
2) To use the decimal notation, a "+" must be prefixed.

Response

Table 62: Response data

Data field	Type	Description	Range/Value
set result	Bool_1	The command was valid and has been executed.	0 failed 1 success

Example

- The map was available and has been acitvated.

Table 63: Response example

Interface	Response example
CoLa-A Deprecated	`sAN LocLoadMapToCache 1`

CoLa2	**ISO 8859-15**
	`ANLocLoadMapToCache 1`
	Hexadecimal
	`414E 4C6F 634C 6F61 644D 6170 546F 4361 6368 6520 01`

REST	`{`
	`  "header": {`
	`  "status": 0,`
	`  "message": "Ok"`
	`  },`
	`  "data": {`
	`  "success": true`
	`  }`
	`}`

Generic (C++)	`response: "{"header":{"status":0,"message":"Ok"},"data":{"success":true}}"`
ROS 1	`success=1`
ROS 2	

Complementary information

Corresponding relevant configuration parameters of the configuration file:

```
map:
  file: test.vmap
```

3.2.5 Command: LocClearMapCache

Overview

Removes the cached maps from the RAM of the localization controller.[1)]

Request

Table 64: Request command

Type	Command
Setter	LocClearMapCache

Table 65: Request data

Data field	Type	Description	Range/Value
-	-	-	-

Example

- Instruct to clear cache.

Table 66: Request example

Interface	Type	Request example
CoLa-A Deprecated	sMN	`sMN LocClearMapCache`
CoLa2	MN	`MNLocClearMapCache`
REST	POST	`http://192.168.0.1/api/LocClearMapCache`
Generic (C++)	gen_srv	`./build/gen_service_call LocClearMapCache POST "{}" -d=2`
ROS 1	srv	`rosservice call  LocClearMapCache "{}"`
ROS 2		`ros2 service call LocClearMapCache sick_lidar_localization/srv/LocClearMapCacheSrv "{}"`

Response

Table 67: Response data

Data field	Type	Description	Range/Value
set result	Bool_1	The command was valid and has been executed.	0 failed 1 success

Example

- The instruction to clear the cache has been passed and acknowledged.

Table 68: Response example

Interface	Response example
CoLa-A Deprecated	`sAN LocClearMapCache 1`

CoLa2	**ISO 8859-15**
	`ANLocClearMapCache 0x01`
	Hexadecimal
	`414E 4C6F 6343 6C65 6172 4D61 7043 6163 6865 2001`

1) When the RAM is freed, depends on a garbage collection process. The RAM is not freed instantly in most cases.

Interface	Response example

REST	<pre>{
	    "header": {
	        "status": 0,
	        "message": "Ok"
	    },
	    "data": {
	        "success": true
	    }
	}</pre>

Generic (C++)	<pre>response: "{"header":
	{"status":0,"message":"Ok"},"data":{"success":true}}"</pre>

ROS 1	<pre>success=1</pre>
ROS 2	

3.2.6 Command: LocGetMap

Overview

Read the current map name. If the resulting map name is empty, no map is activated.

Request

Table 69: Request command

Type	Command
Getter	LocGetMap

Table 70: Request data

Data field	Type	Description	Range/Value
-	-	-	-

Example

- Read map name.

Table 71: Request example

Interface	Type	Request example
CoLa-A Deprecated	sMN	`sMN LocGetMap`
CoLa2	MN	`MNLocGetMap`
REST	POST	`http://192.168.0.1/api/LocGetMap`
Generic (C++)	gen_srv	`./build/gen_service_call LocGetMap POST "{}" -d=2`
ROS 1	srv	`rosservice call LocGetMap "{}"`
ROS 2		`ros2 service call LocGetMap sick_lidar_localization/srv/LocGetMapSrv "{}"`

Response

Table 72: Response data

Data field	Type	Description	Range/Value
mapPath	String	The name of the map.	Max length = 255

Example

- The map name is "mytest.vmap".

Table 73: Response example

Interface	Response example
CoLa-A Deprecated	**Hexadecimal length**[1)] `sAN LocGetMap B mytest.vmap`

CoLa2	**ISO 8859-15**[1)]
	`ANLocGetMap 0x000Bmytest.vmap`
	Hexadecimal[1)]
	`414E 4C6F 6347 6574 4D61 7020 000B 6D79 7465 7374 2E76 6D61 70`

REST	`{ "header": { "status": 0, "message": "Ok" }, "data": { "mapPath": "mytest.vmap" } }`

Interface	Response example

Generic (C++)	`response:`
	`{"header":{"status":0,"message":"Ok"},"data":`
	`{"mapPath":"mytest.vmap"}}`

ROS 1	`mappath: "mytest.vmap"`
ROS 2	

1) The string has a prefix that specifies the length of the string. In this case the character length is 0xB, 11_{dec}.

Complementary information

Corresponding relevant configuration parameters of the configuration file:

```
map:
  file: test.vmap
```

3.3 Initialization

3.3.1 Command: LocInitializeAtPose

Overview

Sets the initial pose and automatically optimizes the pose to match the measured scan data in the loaded map. If the **Localization Status** does not reach at least **Warning** after several automatic optimization loops, the input parameters are set and the calculated optimal pose is discarded.

Prerequisites

- The localization software is in system state **Operation**.

Request

Table 74: Request command

Type	Command
Setter	LocInitializeAtPose

Table 75: Request data

Data field		Type	Description	Range/Value
pose { }				
	x	Int_32	X coordinate	[mm]
	y	Int_32	Y coordinate	[mm]
	yaw	Int_32	Yaw angle	[-180 000,180 000] [mdeg]
searchRadius		UInt_16	Translational uncertainty radius in [mm]. The rotational uncertainty is fixed to ±25°.	[300, 2 000] [mm][1)]

1) Values < 300 will not throw an error, but will internally be set to 300 mm.

Example

Set the pose with the following parameters:

- X coordinate: 10.3 m
- Y coordinate: -5.2 m
- Yaw angle: 30°
- searchRadius: 1 m

Table 76: Request example

Interface	Type	Request example
CoLa-A Deprecated	sMN	`sMN LocInitializeAtPose +10300 -5200 +30000 +1000`

CoLa2	MN	**ISO 8859-15**
		`MNLocInitializeAtPose +10300-5200+30000+1000`
		Hexadecimal
		`4D4E 4C6F 6349 6E69 7469 616C 697A 6541 7450 6F73 6520 0000 0406FF FFEB B000 0075 3003E8`

Interface	Type	Request example

REST	POST	`http://192.168.0.1/api/LocInitializeAtPose`
		`{`
		`"data":`
		`{`
		`"pose":`
		`{`
		`"x": 10300,`
		`"y": -5200,`
		`"yaw": 30000`
		`},`
		`"searchRadius": 1000`
		`}`
		`}`

Generic (C++)	gen_srv	`./build/gen_service_call LocInitializeAtPose POST "{\"data\": {\"pose\":{\"x\":10300, \"y\":-5200, \"yaw\":30000}, \"searchRadius\":1000}}" -d=2`
ROS 1	srv	`rosservice call /LocInitializeAtPose "{x: 10300, y: -5200, yaw: 30000, searchradius: 1000}"`
ROS 2		`ros2 service call /LocInitializeAtPose sick_lidar_localization/srv/LocInitializeAtPoseSrv "{x: 10300, y: -5200, yaw: 30000, searchradius: 1000}"`

Response

Table 77: Response data

Data field	Type	Description	Range/Value
set result	Bool_1	The command was valid and has been executed.	0 failed 1 success

Example

- The initial pose is set and the measured scans could be matched against the map. A failure reason is, for example, if the **Localization Status** did not reach at least **Warning** after several automatic unsuccessful matches.

Table 78: Response example

Interface	Response example
CoLa-A Deprecated	`sAN LocInitializeAtPose 1`

CoLa2	**ISO 8859-15**
	`ANLocInitializeAtPose 1`
	Hexadecimal
	`414E 4C6F 6349 6E69 7469 616C 697A 6541 7450 6F73 6520 01`

REST	`{`
	`"header": {`
	`"status": 0,`
	`"message": "Ok"`
	`},`
	`"data": {`
	`"success": true`
	`}`
	`}`

Generic (C++)	`response: "{"header": {"status":0,"message":"Ok"},"data":{"success":true}}"`

Interface	Response example
ROS 1	`success=1`
ROS 2	

3.3.2 Command: LocResumeAtPose

Overview

Sets the pose to the passed pose parameters. The **Localization Status** is set to **Ok** regardless to the actual map match. It can take between 10 ... 100 ms until the Localization Status switches to Ok.

NOTE
To initialize the vehicle after a restart, the command is not needed when AutoStart is active. The command could be useful in scenarios where you cannot initialize the vehicle, however, the operator is sure where the vehicle is located. In such a scenario, you could not set the vehicle to **Operation** without this command.

Important information

NOTICE
If **LocResumeAtPose** is sent, no scan matching with the map is made. If the sent pose differs from the real pose of the vehicle ("Kidnaped Robot"), accidents can occur.

- Use **LocResumeAtPose** only when you can ensure that the sent pose equals the real pose.

DANGER
Danger of collision
When localizing from the initial position, the vehicle may collide with persons or obstacles.

► Ensure that a safety system is implemented in the vehicle.

Prerequisites

- The localization software is in system state **Operation**.

Request

Table 79: Request command

Type	Command
Setter	LocResumeAtPose

Table 80: Request data

Data field		Type	Description	Range/Value
pose { }				
	x	Int_32	X coordinate	[mm]
	y	Int_32	Y coordinate	[mm]
	yaw	Int_32	Yaw angle	[-180 000,180 000] [mdeg]

Example

Set the pose with the following parameters and set the Localization Status to OK:

- X coordinate: 10.3 m
- Y coordinate: -5.2 m
- Yaw angle: 30 deg

Table 81: Request example

Interface	Type	Request example
CoLa-A Deprecated	sMN	`sMN LocResumeAtPose +10300 -5200 +30000`

Interface	Type	Request example
CoLa2	MN	**ISO 8859-15** MNLocResumeAtPose +10300-5200+30000+1000 **Hexadecimal** 4D4E 4C6F 6352 6573 756D 6541 7450 6F73 6520 0000 0406FF FFEB B000 0075 3003E8

REST	POST	http://192.168.0.1/api/LocResumeAtPose
		{
		"data":
		{
		"pose":
		{
		"x": 10300,
		"y": -5200,
		"yaw": 30000
		},
		}
		}

Generic (C++)	gen_srv	./build/gen_service_call LocResumeAtPose POST "{\"data\": {\"x\":10300, \"y\":5200, \"yaw\":30000}}" -d=2
ROS 1	srv	rosservice call LocResumeAtPose "{x: 10300, y: -5200, yaw: 30000}"
ROS 2		ros2 service call LocResumeAtPose sick_lidar_localization/srv/LocResumeAtPoseSrv "{x: 10300, y: -5200, yaw: 30000}"

Response

Table 82: Response data

Data field	Type	Description	Range/Value
set result	Bool_1	The command was valid and has been executed.	0 failed 1 success

Example

- The initial pose is configured and the **Localization Status** is set to **OK**.

Table 83: Response example

Interface	Response example
CoLa-A Deprecated	sAN LocResumeAtPose 1
CoLa2	**ISO 8859-15** ANLocResumeAtPose 1 **Hexadecimal** 414E 4C6F 6352 6573 756D 6541 7450 6F73 6520 01

REST	{
	"header": {
	"status": 0,
	"message": "Ok"
	},
	"data": {
	"success": true
	}
	}

Generic (C++)	response: "{"header":{"status":0,"message":"Ok"},"data":{"success":true}}"

Interface	Response example
ROS 1	`success=1`
ROS 2	

3.4 Automatic start

3.4.1 Command: LocAutoStartSavePose

Overview

The command is Deprecated. Auto Start does not need commands to be called by the vehicle software. Auto Start can be configured to automatically save poses.

Saves the current pose on-request for the automatic start of the application software.

Important information

NOTICE
The automatic start function might not work correctly if the vehicle is switched off while moving.

NOTICE
The automatic start function does not work correctly if the vehicle is moved after switching it off ("Kidnaped Robot").

► Only place the vehicle at the location where it was switched off or initialize the pose manually.

DANGER
Danger of collision
When localizing from the initial position, the vehicle may collide with persons or obstacles.

► Ensure that a safety system is implemented in the vehicle.

Prerequisites

- The automatic start has to be enabled in the configuration file in section **autoStart**. The **savePoseInterval** should be set to 0 to keep the continuous saving switched off.

```
autoStart:
  active: true          # activate AutoStart [true, false]
  savePoseInterval: 0   # interval in sec to save system state
```

Request

Table 84: Request command

Type	Command
Setter	LocAutoStartSavePose

Table 85: Request data

Data field	Type	Description	Range/Value
-	-	-	-

Example

- Save the current pose for initializing at the next automatic start.

Table 86: Request example

Interface	Type	Request example
CoLa-A Deprecated	sMN	`sMN LocAutoStartSavePose`
CoLa2	MN	`MNLocAutoStartSavePose`
REST	POST	`http://192.168.0.1/api/LocAutoStartSavePose`

Interface	Type	Request example
Generic (C++)	gen_srv	`./build/gen_service_call LocAutoStartSavePose POST "{}" -d=2`
ROS 1	srv	`rosservice call LocAutoStartSavePose "{}"`
ROS 2		`ros2 service call LocAutoStartSavePose sick_lidar_localization/srv/LocAutoStartSavePoseSrv "{}"`

Response

Table 87: Response data

Data field	Type	Description	Range/Value
set result	Bool_1	The command was valid and has been executed.	0 failed 1 success

Example

- The pose has been saved.

Table 88: Response example

Interface	Response example
CoLa-A Deprecated	`sAN LocAutoStartSavePose 1`

CoLa2	**ISO 8859-15**
	`ANLocAutoStartSavePose 0x01`
	Hexadecimal
	`414E 4C6F 6341 7574 6F53 7461 7274 5361 7665 506F 7365 2001`

REST	`{ "header": { "status": 0, "message": "Ok" }, "data": { "success": true } }`
Generic (C++)	`response: "{"header": {"status":0,"message":"Ok"},"data":{"success":true}}"`
ROS 1	`success=1`
ROS 2	

Complementary information

Corresponding relevant configuration parameters of the configuration file:

```
autoStart:
  active: true          # activate AutoStart [true, false]
  savePoseInterval: 0   # interval in sec to save system state
```

3.5 Odometry

3.5.1 Command: LocSetOdometryActive

Overview

Enable or disable the support of encoders or precomputed wheel odometry to enhance the robustness of the contour localization algorithm.

If the wheel odometry is configured in the configuration file, the input is active by default. The **LocSetOdometryActive** command can be used to disable the odometry in certain areas where odometry provides erroneous data.

This command does not effect the output. The LiDAR odometry is still active.

Prerequisites

- The odometer has to be configured in the configuration file in section **vehicle** > **odometer**.
- At least one LiDAR has to be configured and is providing data.

Request

Table 89: Request command

Type	Command
Setter	LocSetOdometryActive

Table 90: Request data

Data field	Type	Description	Range/Value
state	Bool_1	True to activate, false to deactivate the usage odometry for enhanced robustness.	0 false 1 true (default)

Example

- Set the odometry for support active.

Table 91: Request example

Interface	Type	Request example
CoLa-A Deprecated	sMN	`sMN LocSetOdometryActive 1`
CoLa2	MN	`MNLocSetOdometryActive 1`
REST	POST	`http://192.168.0.1/api/LocSetOdometryActive` `{ "data": { "active": true } }`
Generic (C++)	gen_srv	`./build/gen_service_call LocSetOdometryActive POST "{\"data\": {\"active\":true}}" -d=2`
ROS 1	srv	`rosservice call LocSetOdometryActive "{active: true}"`
ROS 2		`ros2 service call LocSetOdometryActive sick_lidar_localization/srv/LocSetOdometryActiveSrv "{active: true}"`

Response

Table 92: Response data

Data field	Type	Description	Range/Value
set result	Bool_1	The command was valid and has been executed.	0 failed 1 success

Example

- The odometry support has been activated.

Table 93: Response example

Interface	Response example
CoLa-A Deprecated	`sAN LocSetOdometryActive 1`

CoLa2	**ISO 8859-15**
	`ANLocSetMappingActive 0x01`
	Hexadecimal
	`414E 4C6F 6353 6574 4F64 6F6D 6574 7279 4163 7469 7665 2001`

REST	`{ "header": { "status": 0, "message": "Ok" }, "data": { "success": true } }`
Generic (C++)	`response: "{"header": {"status":0,"message":"Ok"},"data":{"success":true}}"`
ROS 1	`success=1`
ROS 2	

Complementary information

Corresponding relevant configuration parameters of the configuration file:

```
vehicle:
  odometer:
    type: none      # [none / differentialDrive / external]
    differentialDrive:
       encoderLeft:
        name: left           # identifier for debugging
        radius: 0.1          # wheel radius
        threshold: 1
        type: ABSOLUTE       # [INCREMENTAL / ABSOLUTE]
        pose:                # 2D pose of the encoder
          type: 2D
          x: 0.0
          y: 0.195
          yaw: 0.0
        interface:
          sourceId: 21       # interface id
        ticsPerTurn: -83635  # tics for an entire turn
        maxValue: 4294967295 # ABSOLUTE encoders overflow
      encoderRight:
        name: right          # identifier for debugging
        radius: 0.1          # wheel radius
        threshold: 1
        type: INCREMENTAL    # [INCREMENTAL / ABSOLUTE]
```

```
    pose:                # 2D pose of the encoder
      type: 2D
      x: 0.0
      y: -0.195
      yaw: 0.0
    interface:
      sourceId: 22       # interface id
    ticsPerTurn: 1326.5  # tics for an entire turn
external:
  name: odomIn           # identifier for debugging
  pose:                  # 2D pose of the odometer
    type: 2D
    x: 0
    y: 0
    yaw: 0
  interface:
    sourceId: 31         # interface id
```

3.6 Kinematic

3.6.1 Command: LocSetKinematicVehicleModelActive

Overview

Optional (expert): The kinematics model (odometry and Y-movement restriction) can be switched off for the duration of the recording. Deactivate the kinematics model when the vehicle is moved manually without providing correct wheel odometry values

Request

Table 94: Request command

Type	Command
Setter	LocSetKinematicVehicleModelActive

Table 95: Request data

Data field	Type	Description	Range/Value
state	Bool_1	True to activate, false to deactivate the usage of the kinematic model.	0 false 1 true

Example

- Set the usage of the kinematic vehicle model active.

Table 96: Request example

Interface	Type	Request example
CoLa-A Deprecated	sMN	`sMN LocSetKinematicVehicleModelActive 1`
CoLa2	MN	`MNLocSetKinematicVehicleModelActive 1`
REST	POST	`http://192.168.0.1/api/LocSetKinematicVehicleModelActive` `{ "data": { "active": true } }`
Generic (C++)	gen_srv	`./build/gen_service_call LocSetKinematicVehicleModelActive POST "{\"data\": {\"active\":true}}" -d=2`
ROS 1	srv	`rosservice call LocSetKinematicVehicleModelActive "{active: true}"`
ROS 2		`ros2 service call LocSetKinematicVehicleModelActive sick_lidar_localization/srv/LocSetKinematicVehicleModelActiveSrv "{active: true}"`

Response

Table 97: Response data

Data field	Type	Description	Range/Value
set result	Bool_1	The command was valid and has been executed.	0 failed 1 success

Example

- The kinematic vehicle mode is active.

Table 98: Response example

Interface	Response example
CoLa-A Deprecated	`sAN LocSetKinematicVehicleModelActive 1`

CoLa2	**ISO 8859-15**
	`ANLocSetKinematicVehicleModelActive 0x01`
	Hexadecimal
	`414E 4C6F 6353 6574 4B69 6E65 6D61 7469 6356 6568 6963 6C65 4D6F 6465 6C41 6374 6976 6520 01`

REST	`{ "header": { "status": 0, "message": "Ok" }, "data": { "success": true } }`
Generic (C++)	`response: "{"header": {"status":0,"message":"Ok"},"data":{"success":true}}"`
ROS 1	`success=1`
ROS 2	

Complementary information

The command sets the odometry and restriction of the Y motion temporarily inactive.

Corresponding relevant configuration parameters of the configuration file:

```
vehicle:
   physicalConstraints:
    linearMotion:
      vehicleCannotMoveAlongItsYAxis: false
   odometer:
    type: none
```

3.7 Lines

3.7.1 Command: LocSetLinesForSupportActive

Overview

The command is Deprecated. Lines itself are supported for further versions.

Enable or disable the use of the physical line sensor to support the LiDAR localization.

This command does not effect the output. The virtual line and code output is still active.

Request

Table 99: Request command

Type	Command
Setter	LocSetLinesForSupportActive

Table 100: Request data

Data field	Type	Description	Range/Value
state	Bool_1	True to activate, false to deactivate the usage of line sensors.	0 false 1 true

Example

- Set the line support active.

Table 101: Request example

Interface	Type	Request example
CoLa2	MN	`MNLocSetLinesForSupportActive 1`
REST	POST	`http://192.168.0.1/api/LocSetLinesForSupportActive` `{ "data": { "active": true } }`
Generic (C++)	gen_srv	`./build/gen_service_call LocSetLinesForSupportActive POST "{\"data\": {\"active\":true}}" -d=2`
ROS 1	srv	`rosservice call LocSetLinesForSupportActive "{active: true}"`
ROS 2		`ros2 service call LocSetLinesForSupportActive sick_lidar_localization/srv/LocSetLinesForSupportActiveSrv "{active: true}"`

Response

Table 102: Response data

Data field	Type	Description	Range/Value
set result	Bool_1	The command was valid and has been executed.	0 failed 1 success

Example

- The line support is activated.

Table 103: Response example

Interface	Response example

CoLa2	**ISO 8859-15**
	`ANLocSetLinesForSupportActive 0x01`
	Hexadecimal
	`414E 4C6F 6353 6574 4C69 6E65 7346 6F72 5375 7070 6F72 7441 6374 6976 6520 01`

REST	`{ "header": { "status": 0, "message": "Ok" }, "data": { "success": true } }`
Generic (C++)	`response: "{"header": {"status":0,"message":"Ok"},"data":{"success":true}}"`
ROS 1	`success=1`
ROS 2	

Complementary information

Corresponding relevant configuration parameters of the configuration file:

Input messages

```
sensors:
  line:
    - active: false        # active [true/ false]
      name : mls-0         # sensor identifier
      width: 0.3           # width of the line sensor
      pose:                # 2D mounting pose of the sensor
        type: 2D
        x: 0.652
        y: 0
        yaw: 180
      interface:           # UDP input interface
        sourceId: 1        # identifier
```

Output messages

```
sensors:
  virtualLine:
    - active: false        # active [true/ false]
      name: vlsFront1      # sensor identifier
      width: 0.6           # width of the line sensor
      pose:                # virtual 2D mounting pose
        type: 2D
        x: 0.652
        y: 0
        yaw: 180
      interface:           # UDP output interface
        sourceId: 101      # output message identifier
        msgVersion: 3      # version of the UDP message
```

3.8 Synchronization

3.8.1 Command: LocRequestTimestamp

Overview

Request the current UNIX system time of the localization controller.

Request

Table 104: Request command

Type	Command
Getter	LocRequestTimestamp

Table 105: Request data

Data field	Type	Description	Range/Value
-	-	-	-

Example

- Request the time stamp from the localization controller.

Table 106: Request example

Interface	Type	Request example
CoLa-A Deprecated	sMN	`sMN LocRequestTimestamp`
CoLa2	MN	`MNLocRequestTimestamp`
REST	POST	`http://192.168.0.1/api/LocRequestTimestamp`
Generic (C++)	gen_srv	`./build/gen_service_call LocRequestTimestamp POST "{}" -d=2`
ROS 1	srv	`rosservice call LocRequestTimestamp "{}"`
ROS 2		`ros2 service call LocRequestTimestamp sick_lidar_localization/srv/LocRequestTimestampSrv "{}"`

Response

Table 107: Response data

Data field	Type	Description	Range/Value
Time stamp	UInt_64	The current UNIX system time.	0 ... 1.8E19 in [us]

Example

- The current UNIX system time is 19.909.748.983 us, 19.909.748 ms or UTC time & date: Jan 01 1970 05:31:49

Table 108: Response example

Interface	Response example
CoLa-A Deprecated	`sAN LocRequestTimestamp 19909748983`

CoLa2	**ISO 8859-15**
	`ANLocRequestTimestamp 04 A2 B6 A8 F7`
	Hexadecimal
	`414E 4C6F 6352 6571 7565 7374 5469 6D65 7374 616D 7020 04A2 B6A8 F7`

Interface	Response example

REST	<pre>{
	    "header": {
	        "status": 0,
	        "message": "Ok"
	    },
	    "data": {
	        "timestamp": 19909748983
	    }
	}</pre>

Generic (C++)	<pre>response: "{"header":
	{"status":0,"message":"Ok"},"data":{"timestamp":
	19909748983}}"</pre>

ROS 1	<pre>"timestamp_lidar_us: UInt_64, mean_time_vehicle_us:
	UInt_64, delta_time_us: UInt_64, ..."</pre>

ROS 2	

3.9 Recording

3.9.1 Command: LocSetRecordingActive

Overview

Start or stop the recording of support data for improved support in critical situations.

Request

Table 109: Request command

Type	Command
Setter	LocSetRecordingActive

Table 110: Request data

Data field	Type	Description	Range/Value
state	Bool_1	True to activate, false to deactivate the configuration.	0 false 1 true

Example

- Start the recording of support data.

Table 111: Request example

Interface	Type	Request example
CoLa-A Deprecated	sMN	`sMN LocSetRecordingActive 1`
CoLa2	MN	`MNLocSetRecordingActive 1`

REST	POST	`http://192.168.0.1/api/LocSetRecordingActive`
		`{`
		`"data":`
		`{`
		`"active": true`
		`}`
		`}`

Generic (C++)	gen_srv	`./build/gen_service_call LocSetRecordingActive POST "{\"data\": {\"active\":true}}" -d=2`
ROS 1	srv	`rosservice call LocSetRecordingActive "{active: true}"`
ROS 2		`ros2 service call LocSetRecordingActive sick_lidar_localization/srv/LocSetRecordingActiveSrv "{active: true}"`

Response

Table 112: Response data

Data field	Type	Description	Range/Value
set result	Bool_1	The command was valid and has been executed.	0 failed 1 success

Example

- The recording configuration has been set.

Table 113: Response example

Interface	Response example
CoLa-A Deprecated	`sAN LocSetRecordingActive 1`

CoLa2	**ISO 8859-15**
	`ANLocSetRecordingActive 0x01`
	Hexadecimal
	`414E 4C6F 6353 6574 5265 636F 7264 696E 6741 6374 6976 6520 01`

REST	`{`
	`"header": {`
	`"status": 0,`
	`"message": "Ok"`
	`},`
	`"data": {`
	`"success": true`
	`}`
	`}`

Generic (C++)	`response: "{"header": {"status":0,"message":"Ok"},"data":{"success":true}}"`
ROS 1	`success=1`
ROS 2	

3.9.2 Command: LocSetRingBufferRecordingActive

Overview

Enable or disable the continuous recording of data for improved support in critical situations.

Request

Table 114: Request command

Type	Command
Setter	LocSetRingBufferRecordingActive

Table 115: Request data

Data field	Type	Description	Range/Value
state	Bool_1	True to activate, false to deactivate the configuration.	0 false 1 true

Example

- Set the ring buffer function active.

Table 116: Request example

Interface	Type	Request example
CoLa-A Deprecated	sMN	`sMN LocSetRingBufferRecordingActive 1`
CoLa2	MN	`MNLocSetRingBufferRecordingActive 1`
REST	POST	`http://192.168.0.1/api/LocSetRingBufferRecordingActive` `{ "data": { "active": true } }`
Generic (C++)	gen_srv	`./build/gen_service_call LocSetRingBufferRecordingActive POST "{\"data\": {\"active\":true}}" -d=2`
ROS 1	srv	`rosservice call LocSetRingBufferRecordingActive "{active: true}"`
ROS 2		`ros2 service call LocSetRingBufferRecordingActive sick_lidar_localization/srv/LocSetRingBufferRecordingActiveSrv "{active: true}"`

Response

Table 117: Response data

Data field	Type	Description	Range/Value
set result	Bool_1	The command was valid and has been executed.	0 failed 1 success

Example

- The ring buffer configuration has been set.

Table 118: Response example

Interface	Response example
CoLa-A Deprecated	`sAN LocSetRingBufferRecordingActive 1`

CoLa2	**ISO 8859-15**
	`ANLocSetRingBufferRecordingActive 0x01`
	Hexadecimal
	`414E 4C6F 6353 6574 5269 6E67 4275 6666 6572 5265 636F 7264 696E 6741 6374 6976 6520 01`

REST	`{ "header": { "status": 0, "message": "Ok" }, "data": { "success": true } }`
Generic (C++)	`response: "{"header": {"status":0,"message":"Ok"},"data":{"success":true}}"`
ROS 1	`success=1`
ROS 2	

3.9.3 Command: LocSaveRingBufferRecording

Overview

Saves the most recent data (up to 60 seconds) from the continuous ring buffer recording.

NOTE
If the storage of the sd card is full, the oldest recording file will be deleted when **LocSaveRingBufferRecording** is called

NOTE
The ring buffer recording must be enabled only once by the command **LocSetRingBufferRecordingActive** to use this function.

Request

Table 119: Request command

Type	Command
Setter	LocSaveRingBufferRecording

Table 120: Request data

Data field	Type	Description	Range/Value
reason	String	Reason why the recording should be saved.	Max length = 512

Example

- Saves most recent data with information about time due to error with name “connection error”

Table 121: Request example

Interface	Type	Request example

CoLa-A
Deprecated
	sMN	**Hexadecimal length**[1)]
		`sMN LocSaveRingBufferRecording 24 YYYY-MM-DD_HH-MM-SS connection error`
		Decimal length[1)2)]
		`sMN LocSaveRingBufferRecording +36 YYYY-MM-DD_HH-MM-SS connection error`

CoLa2	MN	**ISO 8859-15**[1)]
		`MNLocSaveRingBufferRecording 0x0024YYYY-MM-DD_HH-MM-SS connection error`
		Hexadecimal
		`4D4E 4C6F 6353 6176 6552 696E 6742 7566 6665 7252 6563 6F72 6469 6E67 2000 2459 5959 592D 4D4D 2D44 445F 4848 2D4D 4D2D 5353 2063 6F6E 6E65 6374 696F 6E20 6572 726F 72`

REST	POST	`http://192.168.0.1/api/LocSaveRingBufferRecording`
		`{`
		`"data":`
		`{`
		`"reason": "YYYY-MM-DD_HH-MM-SS connection error"`
		`}`
		`}`

Interface	Type	Request example
Generic (C++)	gen_srv	`./build/gen_service_call LocSaveRingBufferRecording POST "{\"data\": {\"reason\": \"YYYY-MM-DD_HH-MM-SS connection error\"}}" -d=2`
ROS 1	srv	`rosservice call LocSaveRingBufferRecording "{reason: YYYY-MM-DD_HH-MM-SS connection error}"`
ROS 2		`ros2 service call LocSaveRingBufferRecording sick_lidar_localization/srv/LocSaveRingBufferRecordingSrv "{reason: YYYY-MM-DD_HH-MM-SS connection error}"`

1) The string has a prefix that specifies the length of the string. In this case the character length is 0x24, 36_{dec}.
2) To use the decimal notation, a "+" must be prefixed.

Response

Table 122: Response data

Data field	Type	Description	Range/Value
set result	Bool_1	The command was valid and has been executed.	0 failed 1 success

Example

- The ring buffer recording has been saved. In case of a **failed** response, the SD card might have been full or the ring buffer was not activated.

Table 123: Response example

Interface	Response example
CoLa-A Deprecated	`sAN LocSaveRingBufferRecording 1`

CoLa2	**ISO 8859-15**
	`ANLocSaveRingBufferRecording 1`
	Hexadecimal
	`414E 4C6F 6353 6176 6552 696E 6742 7566 6665 7252 6563 6F72 6469 6E67 2001`

REST	`{ "header": { "status": 0, "message": "Ok" }, "data": { "success": true } }`
Generic (C++)	`response: "{"header": {"status":0,"message":"Ok"},"data":{"success":true}}"`
ROS 1	`success=1`
ROS 2	

4 Streaming API (UDP)

4.1 User interface UDP

Overview

With the streaming interface, the localization controller receives and sends data continuously to the external vehicle controller. For that purpose the binary UDP protocol is used.

The UDP interface can be configured via the configuration file. The parameters are listed in Complementary information.

UDP message types

The following message types can be used in LiDAR-LOC:

Table 124: Streaming UDP message types

MsgType #	Message	Streaming
1	Odometry	Input and output
2	Encoder	Input
3	Codes	Input and output
4	Lines	Input and output
5	Localization results	Output
7	2D codes	Input

Ports

- By default, 5009 is used to receive incoming UDP messages.
  `communication > smopUdp > input >` **port**
- By default, 5008 is used as source port to send outgoing UDP messages.
  `communication > smopUdp > output >` **sourcePort**
- The `localhost` with `127.0.0.1` is used as the destination IP by default. The IP specifies where the UDP messages are sent to in the receiving device (vehicle controller). Only one receiver address can be assigned. Multicast or broadcast addresses can be used for multiple receivers.
  `communication > smopUdp > output >` **destinationIp**
- By default, 5010 is used as the destination port. The port specifies where the UDP messages are sent to in the receiving device (vehicle controller). Only one destination port can be assigned. For different destination ports, e.g. on the same receiving device, port forwarding must be set up on the receiving device.
  `communication > smopUdp > output >` **destinationPort**

Endianness

The endianness of the input header is automatically detected based on the header field **PayloadType**. However, the payload is parsed according to the endianness given by the field PayloadType. The endianness of the output header and payload is BIG_ENDIAN by default and can be changed in the configuration file with the variable:

```
communication > smopUdp > output > endianess
```

Framing

The telegrams consist of the header and the payload. The payload is directly attached to the header.

Table 125: LiDAR-LOC message framing

Section	Description	Size [byte]
Header	The header of each UDP message specifies how the payload is interpreted.	16
Payload	The payload is the part of transmitted data that is the actual intended message.	Depends on MsgType

Complementary information

Corresponding relevant configuration parameters of the configuration file:

Input messages

```
vehicle:
  odometer:
     encoderLeft:
      interface:
        sourceId: 21
       encoderRight:
      interface:
        sourceId: 22
     external:
      interface:
        sourceId: 31
communication:
  smopUdp:
    input:
      port: 5009                # port for input UDP stream
sensors:
  line:
    interface:                  # UDP input interface
      sourceId: 1               # identifier
  codeReader
    interface:                  # UDP input interface
      sourceId: 11              # identifier
   lidar:
    interface:                  # TCP/IP interface
      ipAddress: 192.168.0.41   # IP Address of the sensor
      port: 2122                # sensor port (2111 or 2122)
      receiverPort: 56661       # For MICS3/NANS3 devices
      receiverIP: 192.168.0.11  # IP routing for MICS3/NANS3
```

Output messages

```
communication:
  smopUdp:
    output:
      sourcePort: 5008          # port for sending output
      destinationIp: 127.0.0.1  # address to receive output
      destinationPort: 5010     # port to receive output
      endianess: BIG_ENDIAN
      localizationMsg:
        sourceId: 1             # output message identifier
        msgVersion: 2           # loc. message version
      odometryMsg:
        sourceId: 1             # output message identifier
        msgVersion: 5           # odometry message version
sensors:
  virtualLine
    interface:                  # UDP input interface
      sourceId: 101             # output message identifier
      msgVersion: 2             # output message version
  virtualCodeReader
    interface:                  # UDP input interface
```

```
sourceId: 111            # output message identifier
msgVersion: 4            # output message version
```

4.2 Header

The header of each UDP message specifies how the payload is interpreted.

Table 126: Header format

Variable	Description	Type	Range of Value (decimal)	Size [byte]
MagicWord	**MO**bile **P**latform**S**	String	Fixed: 0x4d4f5053, ("MOPS")	4
HeaderVersion	Version of the header.	UInt16	0 ... 65 535, default: 2	2
PayloadLength	Number of bytes of the payload.	UInt16	0 ... UDP IPV4: 65 491 UDP IPV6: 65 471	2
PayloadType (endianness)[2)]	Configuration of the byte endianness for the payload.	UInt16	BE: 0x0642 (default) LE: 0x06c2	2
MsgType[1)2)]	Definition of the payload type of the message.	UInt16	0 ... 65 535	2
MsgTypeVersion[2)]	Version of the MsgType payload.	UInt16	0 ... 65 535	2
SourceID[2)]	Identification of multiple sources for the same message.	UInt16	0 ... 65 535	2
			HeaderLength	16 byte

1)
- MsgType 1: odometry messages
- MsgType 2: encoder messages
- MsgType 3: 1D code messages
- MsgType 4: line messages
- MsgType 5: localization result messages
- MsgType 7: 2D code messages

2) For input messages, the variable value is specified by the vehicle controller. For output messages, the variable value is specified in the configuration file (*.yml).

Example (hex):

- `4D4F 5053 0002 0018 0642 0001 0004 001F`

Explanation of example:

- MagicWord: 0x4D4F 5053 ("MOPS")
- HeaderVersion: 0x0002
- PayloadLength: 0x0018 (8+8+2+2+4 = 24_{dec})
- PayloadType: 0x0642 (Big Endian)
- MsgType: 1 (odometry)
- MsgTypeVersion: 4 (version 4, velocity-based odometry)
- SourceID: 1F (e. g. odometry sender with no 31_{dec})

4.3 Payload

Overview

The payload is the part of transmitted data that is the actual intended message. The payload is described for **input** and **output** messages. The endianness of the payload can be configured within the header via `PayloadType`.

Table 127: Standard variables

Variable	Description
TelegramCount	The counter starts at 0 and should be increased by 1 for each new message. An overflow (reset) of this variable is handled by the software. **Example** • Message 1: TelegramCount = 0 • Message 2: TelegramCount = 1 • Message 3: TelegramCount = 2 • ...
Time stamp	The time stamp corresponds to the current time when the data is generated in microseconds. • For input messages, the time stamp is generated from the sensor. • For output messages, the time stamp is generated from the localization controller. The time stamp is used to calculate the time difference between two messages. A synchronization between vehicle and localization controller is not required. The SICK Software handles overflow (reset) of the variable. **Example (time passed since start of the controller)** • Message 1: Time stamp = 50 000 • Message 2: Time stamp = 50 025 • Message 3: Time stamp = 50 050 • ...

4.3.1 Odometry - MsgType: 1

Overview

This message can be used to send a precomputed motion of the vehicle to the LiDAR-LOC system.

Using this message does not require to specify any motion model. However, it does require the computation of the motion of the vehicle by the user.

LiDAR-LOC supports two different protocol versions for odometry input:

- MsgTypeVersion 4 contains the velocity vector of the vehicle with v_x, v_y and ω.
- MsgTypeVersion 5 accepts the pose of the vehicle computed by the wheel odometry in absolute values.

We recommend using version 5 to allow LiDAR-LOC to handle lost messages better than with version 4.

NOTE
If the vehicle kinematics is a differential drive, the simplified direct encoder input is preferred.

NOTE
The messages should be sent continuously, with a constant interval of 25 ms ± 20 %. Variations in frequency can be compensated for with a sufficiently accurate time stamp in the message to the localization controller.

Input messages: wheel odometry

The messages are 64-bit aligned.

Table 128: MsgType: 1, Version: 4

Variable	Type	Range of Value (decimal)	Size [byte]
TelegramCount	UInt64	0 ... 1.8E19	8

Variable	Type	Range of Value (decimal)	Size [byte]
Time stamp	UInt64	0 ... 1.8E19 us	8
X-component of velocity	Int16	±32 768 mm/s	2
Y-component of velocity	Int16	±32 768 mm/s	2
Angular velocity	Int32	±2.1E9 mdeg/s	4
		PayloadLength	24_{dec}, 0x18

Table 129: Header and Payload example for MsgType: 1, Version: 4

```
--> 4D4F 5053 0002 0018 0642 0001 0004 001F 0000 0000 0000
03E80000 0000 0000 C350 FE0C01F4 FFFF FC18
```

Explanation of example:

- Header: 4D4F 5053 0002 0018 0642 0001 0004 001F
  - MagicWord: 0x4D4F 5053 ("MOPS")
  - HeaderVersion: 0x0002
  - PayloadLength: 0x0018 (8+8+2+2+4 = 24_{dec})
  - PayloadType: 0x0642 (Big Endian)
  - MsgType: 1 (odometry)
  - MsgTypeVersion: 4 (version 4, velocity-based odometry)
  - SourceID: 1F (e. g. odometry sender with no 31_{dec})
- TelegramCount: 1 000_{dec}, 0x03E8
- time stamp: 50 000 us, 0xC350
- X-component of velocity: -500 mm/s, 0xFE0C
- Y-component of velocity: 500 mm/s, 0x01F4
- Angular velocity: -1 000 mdeg/s, 0xFC18

Table 130: MsgType: 1, Version: 5

Variable	Type	Range of Value (decimal)	Size [byte]
TelegramCount	UInt64	0 ... 1.8E19	8
Time stamp	UInt64	0 ... 1.8E19 us	8
X position[1)]	Int64	±9.2E18 mm	8
Y position[1)]	Int64	±9.2E18 mm	8
yaw[1)]	Int64	±180 000 mdeg	8
		PayloadLength	40_{dec}, 0x28

1) Pose of the vehicle computed by the wheel odometry in absolute values.

Output messages: LiDAR odometry (expert function)

In rare cases, the output from the LiDAR localization might be needed as velocity or incremental position information. This output is the so called LiDAR odometry. The odometry output is the motion of the vehicle before applying the pose correction using the map. Therefore, the LiDAR odometry output is continuous without jumps as the Localization Result (MsgType 5) which is output after the reference scan matching.

Complementary information

Corresponding command:

- LocSetOdometryActive

Corresponding relevant configuration parameters of the configuration file:

Input messages (wheel odometry)

```
vehicle:
  odometer:
    type: external  # [none / differentialDrive / ...]
    external:
      name: odomIn           # identifier for debugging
```

```
      pose:                    # 2D pose of the odometer
        type: 2D
        x: 0
        y: 0
        yaw: 0
      interface:
        sourceId: 31           # interface id
```

Expert: Output messages (LiDAR odometry)

```
  communication:
    smopUdp:                 # UDP input / output configuration
      output:
        odometryMsg:
          sourceId: 1        # output message identifier
          msgVersion: 5      # odometry UDP message version
```

Further topics

- LiDAR-LOC Operating Instructions, 8027121
  - "Wheel odometry and encoder (input)"

4.3.2 Encoder - MsgType: 2

Overview

This message is used to send encoder measurements to the LiDAR-LOC system such that LiDAR-LOC computes the motion of the vehicle using the specified motion model. LiDAR-LOC supports direct wheel encoder input only for differential drives. In the configuration file: `vehicle > odometer > type: differentialDrive`.

Implementation

The messages should be sent continuously, with a constant interval of 25 ms ± 20 %. Variations in frequency can be compensated for with a sufficiently accurate time stamp in the message to the localization controller.

The Encoder messages for left and right wheel should be sent at the same time. The break between two consecutive messages for each left an right message being around 25 ms.

Each input message for left and right wheel needs to be sent in a full message including header and payload. The messages for left and right wheel cannot be combined.

Prerequisites

- Specified motion model in the configuration file.

Input messages

The messages are 64-bit aligned.

Table 131: MsgType: 2, Version: 2

Variable	Type	Range of Value (decimal)	Size [byte]
TelegramCount	UInt64	0 ... 1.8E19	8
Time stamp	UInt64	0 ... 1.8E19 us	8
Encoder value[1)]	Int64	±9.2E18	8
		PayloadLength	24_{dec}, 0x18

1) In tics, either absolute or incremental

Table 132: Header and Payload example for MsgType: 2, Version: 2

```
--> 4D4F 5053 0002 0018 0642 0002 0002 0015 0000 0000 0000
03E80000 0000 0000 C350 0000 0000 0000 3039
```

Explanation of example:

- Header: 4D4F 5053 0002 0018 0642 0002 0002 0015
  - MagicWord: 0x4D4F 5053 ("MOPS")
  - HeaderVersion: 0x0002
  - PayloadLength: 0x0018 (8+8+2+2+4 = 24_{dec})
  - PayloadType: 0x0642 (Big Endian)
  - MsgType: 1 (Encoder)
  - MsgTypeVersion: 2
  - SourceID: 15 (e. g. left encoder with no 21_{dec})
- TelegramCount: $1\,000_{dec}$, 0x03E8
- Time stamp: 50 000 us, 0xC350
- Encoder value: 12345, 0x3039

Complementary information

Corresponding command:

- LocSetOdometryActive

Corresponding relevant configuration parameters of the configuration file:

Input messages

```
vehicle:
  odometer:
    type: differentialDrive  # [none / ... / external]
    differentialDrive:
       encoderLeft:
        name: left           # identifier for debugging
        radius: 0.1          # wheel radius
        threshold: 1
        type: ABSOLUTE       # [INCREMENTAL / ABSOLUTE]
        pose:                # 2D pose of the encoder
          type: 2D
          x: 0.0
          y: 0.195
          yaw: 0.0
        interface:
          sourceId: 21       # interface id
        ticsPerTurn: -83635  # tics for an entire turn
        maxValue: 4294967295 # ABSOLUTE encoders overflow
      encoderRight:
        name: right          # identifier for debugging
        radius: 0.1          # wheel radius
        threshold: 1
        type: INCREMENTAL    # [INCREMENTAL / ABSOLUTE]
        pose:                # 2D pose of the encoder
          type: 2D
          x: 0.0
          y: -0.195
          yaw: 0.0
        interface:
          sourceId: 22       # interface id
        ticsPerTurn: 1326.5  # tics for an entire turn
```

Further topics

- LiDAR-LOC Operating Instructions, 8027121
  - "Wheel odometry and encoder (input)"

4.3.3 1D Codes - MsgType: 3

Overview

Input messages to send event-based code measurements from vehicle controller to localization device when a code is detected. The map does not have to contain the code to be recognized as an input message.

Output messages from localization device to vehicle controller to send virtual code measurements from the LiDAR-LOC.

Prerequisites

- For output messages: The codes are specified in the map.

Input messages

Table 133: MsgType: 3, Version: 3

Variable	Type	Range of Value (decimal)	Size [byte]
TelegramCount	UInt64	0 ... 1.8E19	8
Time stamp	UInt64	0 ... 1.8E19 us	8
Code	Int64	±9.2E18	8
		PayloadLength	24_{dec}, 0x18

The messages are 64-bit aligned.

Payload example (hex):

- 0000 0000 0000 03E8 0000 0000 0000 C350 0000 0000 0000 3039

Explanation of example:

- TelegramCount: $1\,000_{dec}$, 0x03E8
- Time stamp: 50 000 us, 0xC350
- Code value: 12345, 0x3039

Table 134: MsgType: 3, Version: 5

Variable	Type	Range of Value (decimal)	Size [byte]
TelegramCount	UInt64	0 ... 1.8E19	8
Time stamp	UInt64	0 ... 1.8E19 us	8
CodeLength	UInt16	0 ... 65 535	2
Code	String	-	CodeLength
		PayloadLength	24_{dec} +Code-Length, 0x18 +Code-Length

Due to the dynamic size of MsgType 3, version 5, the message is not memory aligned.

Output messages

Table 135: MsgType: 3, Version: 4

Variable	Type	Range of Value (decimal)	Size [byte]
TelegramCount	UInt64	0 ... 1.8E19	8
Time stamp	UInt64	0 ... 1.8E19 us	8
Code	Int32	±2.1E9	4
Distance[1)]	Int32	±2.1E9 mm	4
		PayloadLength	24_{dec}, 0x18

1) Distance from center of sensor.

The messages are 64-bit aligned.

Payload example (hex):

- 0000 0000 0000 03E8 0000 0000 0000 C350 0000 3039 0000 FFE2

Explanation of example:

- TelegramCount: 1 000_{dec}, 0x03E8
- Time stamp: 50 000 us, 0xC350
- Code value: 12345, 0x3039
- Distance: -30 mm, 0xFFE2

Complementary information

Corresponding relevant configuration parameters of the configuration file:

Input messages

```
sensors:
  codeReader:
    - name: rfidReader          # sensor identifier
      active: true              # active [true/ false]
      pose:                     # mounting pose of the sensor
        type: 2D
        x: 0.573
        y: 0.118
        yaw: 0
      interface:                 # UDP input interface
        sourceId: 11             # identifier
```

Output messages

```
sensors:
  virtualCodeReader:
    - active: false              # active [true/ false]
      name: vcsRadial            # sensor identifier
      type: RADIAL               # options are [LINE / RADIAL]
      size: 0.2                  # reading area diameter
      pose:                      # virtual 2D mounting pose
        type: 2D
        x: 0.573
        y: 0.118
        yaw: 0
      interface:                 # UDP output interface
        sourceId: 111            # output message identifier
        msgVersion: 4            # version of the UDP message
    - active: false              # active [true/ false]
      name: vcsLine              # sensor identifier
      type: LINE                 # options are [LINE / RADIAL]
      size: 0.3                  # virtual sensor line length
      pose:                      # virtual 2D mounting pose
        type: 2D
        x: 0.573
        y: 0.15
        yaw: 0
      interface:                 # UDP output interface
        sourceId: 112            # output message identifier
        msgVersion: 4            # version of the UDP message
```

Further topics

- LiDAR-LOC Operating Instructions, 8027121
  - "Code sensors"

4.3.4 Line - MsgType: 4

Overview

This message can be used to send line measurements from a physical line sensor to the LiDAR-LOC System or to receive virtual line measurement from the LiDAR-LOC System.

LiDAR-LOC supports two different protocol versions:

- Version 3 allows a variable number of lanes
- Version 4 has a fixed size for up to 3 lanes

For the functionality of LiDAR-LOC both are equivalent.

Input and output messages

Table 136: MsgType: 4, Version: 3

Variable	Type	Range of Value (decimal)	Size [byte]
TelegramCount	UInt64	0 ... 1.8E19	8
Time stamp	UInt64	0 ... 1.8E19 us	8
NumOfLinesVisible	Unt8	255	1
Line1[1)]	Int16	±32 768 mm	2
Line2[1)]	Int16	±32 768 mm	2
Line3[1)]	Int16	±32 768 mm	2
...	Int16	±32 768 mm	...
		PayloadLength	17 + $2n_{dec}$

1) Exists only if corresponding NumOfLinesVisible >= Line#
Line1 contains the lowest value, increasing with the line number.

Due to the dynamic size of MsgType: 4, Version: 3, the message is not memory aligned.

Payload example for message version 3 (hex):

- 0000 0000 0000 03E8 0000 0000 0000 C350 02 0032 FFEC

Explanation of example:

- TelegramCount: 1 000_{dec}, 0x03E8
- time stamp: 50 000 us, 0xC350
- NumOfLinesVisible: 2, 0x02
- Line1: 50 mm, 0x0032
- Line2: -20 mm, 0xFFEC

Table 137: MsgType: 4, Version: 4

Variable	Type	Range of Value (decimal)	Size [byte]
TelegramCount	UInt64	0 ... 1.8E19	8
Time stamp	UInt64	0 ... 1.8E19 us	8
LCP1[1)]	Int16	±32 768 mm	2
LCP2[1)]	Int16	±32 768 mm	2
LCP3[1)]	Int16	±32 768 mm	2
#LCP[2)]	UInt8	0 ... 7	1
Reserved	UInt8	-	1

Variable	Type	Range of Value (decimal)	Size [byte]
		PayloadLength	24_{dec}, 0x18

1) Line Center Point: Line offset to sensor center in mm
The principle of LCP1 < LCP2 < LCP3 always applies to the LCPs.

2) The field #LCP can take the following values:
- 0: No lane visible. Ignore all values from LCP1-LCP3.
- 2: Exactly one lane visible and stored in LCP2 (one middle lane). Ignore the values from LCP1 and LCP3.
- 3: Exactly two lanes visible. Lowest value stored in LCP1 with higher value stored in LCP2 (one left and one middle lane). Ignore the values from LCP3.
- 6: Exactly two lanes visible. Lowest value stored in LCP2 with higher value in LCP3 (one middle and one right lane). Ignore the values from LCP1.
- 7: Exactly three lanes visible. Lowest value stored in LCP1 with higher values in LCP2 and LCP3 respectivley.

Payload example for message version 4 (hex):
- 0000 0000 0000 03E8 0000 0000 0000 C350 FF95 0032 FF95 0300

Explanation of example:
- TelegramCount: $1\,000_{dec}$, 0x03E8
- time stamp: 50 000 us, 0xC350
- LCP1: -107 mm, 0xFF95
- LCP2: 50 mm, 0x0032
- LCP3: -107 mm, 0xFF95
- #LCP: 3, 0x03
- Reserved: 0x00

The messages are 64-bit aligned.

Complementary information

Corresponding relevant configuration parameters of the configuration file:

Input messages

```
sensors:
  line:
    - active: false       # active [true/ false]
      name : mls-0        # sensor identifier
      width: 0.3          # width of the line sensor
      pose:               # 2D mounting pose of the sensor
        type: 2D
        x: 0.652
        y: 0
        yaw: 180
      interface:          # UDP input interface
        sourceId: 1       # identifier
```

Output messages

```
sensors:
  virtualLine:
    - active: false       # active [true/ false]
      name: vlsFront1     # sensor identifier
      width: 0.6          # width of the line sensor
      pose:               # virtual 2D mounting pose
        type: 2D
        x: 0.652
        y: 0
        yaw: 180
      interface:          # UDP output interface
        sourceId: 101     # output message identifier
        msgVersion: 3     # version of the UDP message
```

Further topics

- LiDAR-LOC Operating Instructions, 8027121
  - "Line sensors"

4.3.5 Localization result - MsgType: 5

Overview

This message is used to receive localization results from the LiDAR-LOC system.

Prerequisites

- The localization software is in system state **Operation**.

Output messages

Table 138: MsgType: 5, Version: 2

Variable	Type	Range of Value (decimal)	Size [byte]
TelegramCount	UInt64	0 ... 1.8E19	8
Time stamp	UInt64	0 ... 1.8E19 us	8
X	Int64	±9.2E18 mm	8
Y	Int64	±9.2E18 mm	8
Yaw	Int32	±180 000 mdeg	4
LocalizationStatus[1)]	UInt8	10: OK 20: Warning 30: Not Localized 40: System Error 100: Undefined Return value 100 when system state is in **Configuration**.	1
MapMatchingStatus[2)]	UInt8	0: None 30: Low 60: Medium 90: Good	1
Reserved	UInt16	-	2
		PayloadLength	40_{dec}, 0x28

1) The localization status indicates the stability and reliability of the current localization. The status is the value for controlling the vehicle movement during operation.

2) The Map Matching Status indicates how often and how well the system has corrected the pose by matching scans to the map. How well the environment is suited for LiDAR localization at the current pose. How well the current scans match the map -> how up-to-date the map is. The status can be regarded as a process evaluation and monitoring parameter. This enables the map to be evaluated, and corrective measures to improve the map quality to be identified.

The messages are 64-bit aligned.

Payload example (hex):

- 0000 0000 0000 03E8 0000 0000 0000 C350 0000 0000 0000 005D 0000 0000 0000 0021 0000 45E7 0A5A 0000

Explanation of example:

- TelegramCount: $1\,000_{dec}$, 0x03E8
- Time stamp: 50 000 us, 0xC350
- X: 93 mm, 0x005D
- Y: 33 mm, 0x0021
- Yaw: 17 895 mdeg, 0x45E7
- LocalizationStatus: 10 (OK), 0x0A

- MapMatchingStatus: 90 (Good), 0x5A
- Reserved: 0, 0x0

Table 139: MsgType: 5, Version: 3

Variable	Type	Range of Value (decimal)	Size [byte]
TelegramCount	UInt64	0 ... 1.8E19	8
Time stamp	UInt64	0 ... 1.8E19 us	8
X	Int64	±9.2E18 mm	8
Y	Int64	±9.2E18 mm	8
Yaw	Int32	±180 000 mdeg	4
LocalizationStatus	UInt8	10: OK 20: Warning 30: Not Localized 40: System Error 100: Undefined Return value 100 when system state is in **Configuration**.	1
MapMatchingStatus	UInt8	0: None 30: Low 60: Medium 90: Good	1
MapNameLength	UInt16	0 ... 255	2
MapName	String	-	MapName-Length
		PayloadLength	40_{dec}+Map-Name-Length, 0x28 +Map-NameLength

Due to the dynamic size of MsgType 5, version 3, the message is not memory aligned.

Complementary information

Corresponding relevant configuration parameters of the configuration file:

Output messages

```
communication:
  smopUdp:
    output:
      localizationMsg:
        sourceId: 1             # output message identifier
        msgVersion: 2           # UDP message version
```

4.3.6 2D Codes - MsgType: 7

Overview

This message is used to send event-based 2D code measurements to the LiDAR-LOC system when a 2D code sensor reads a code.

Prerequisites

- The codes are specified in the map.

Input messages

Table 140: MsgType: 7 Version: 1

Variable	Type	Range of Value (decimal)	Size [byte]
TelegramCount	UInt64	0 ... 1.8E19	8
Time stamp	UInt64	0 ... 1.8E19 us	8
CodeLength	UInt16	0 ... 65 535	2
Code	String	-	CodeLength
X code position[1)]	Int32	±2.1E9 mm	4
Y code position[1)]	Int32	±2.1E9 mm	4
Yaw code orientation[1)]	Int32	±180 000 mdeg	4
		PayloadLength	30_{dec} + CodeLength 0x1E + CodeLength

1) Relative pose of the code along the according axis in the sensor frame.

Table 141: Header and Payload example for MsgType: 7, Version: 1

```
--> 4D4F 5053 0002 0022 0642 0007 0001 000B  0000 0000 0000 03E8
0000 0000 0000 C350 00043130 3130 0000 005D 0000 0021 0000 05AA
```

Explanation of example:

- Header: 4D4F 5053 0002 0022 0642 0007 0001 000B
  - MagicWord: 0x4D4F 5053 ("MOPS")
  - HeaderVersion: 0x0002
  - PayloadLength: 0x0022 (34_{dec})
  - PayloadType: 0x0642 (Big Endian)
  - MsgType: 7 (2D Codes)
  - MsgTypeVersion: 1 (version 1)
  - SourceID: 0x0B (e. g. 2D code sender with no 11_{dec})
- TelegramCount: $1\,000_{dec}$, 0x03E8
- Time stamp: 50 000 us, 0xC350
- CodeLength: 4 byte, 0x04
- Code: "1010", 0x31303130
- X: 93 mm, 0x005D
- Y: 23 mm, 0x0021
- Yaw: 1 450 mdeg, 0x05AA

Complementary information

Corresponding relevant configuration parameters of the configuration file:

Input messages

```
sensors:
  code2DReader:
    - name: code2DReader_1     # sensor identifier
      active: true             # active [true/ false]
      interfaceType: MOPS_UDP # Type [MOPS_UDP /
SENSOR_ETHERNET]
        mopsinterface:         # UDP input interface
        sourceId: 11           # identifier
         timeOffset: 0
       pose:                   # mounting pose of the sensor
        type: 2D
        x: 0.573
        y: 0.118
        yaw: 0
```

Further topics

- LiDAR-LOC Operating Instructions, 8027121
  - "Code sensors"

5 Annex

5.1 Appendix A: Communication via CoLa2

Overview

CoLa2 (Command Language 2) is a protocol from SICK, with which a client (controller, computer etc.) can access suitable SICK sensors and sofware via a network (TCP/IP).

The CoLa2 interface of the software allows you to request information from the software (read a parameter) or configure the software (set a parameter).

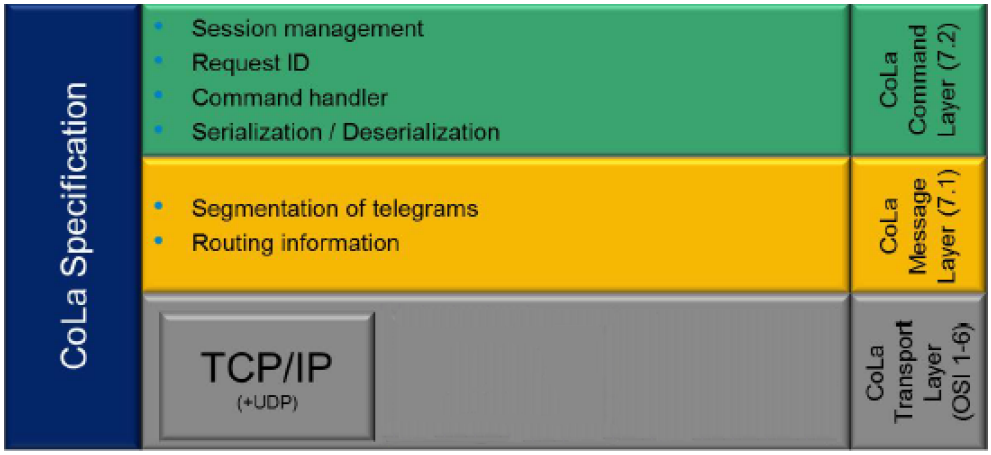

Figure 5: CoLa2 protocol stack

How a client (vehicle controller, computers etc.) communicates with a server (LiDAR-LOC) via CoLa2 is described in the following.

In this context, every IT device counts as a client, for which the following applies:

- The device accesses LiDAR-LOC.
- The device sends data or commands to LiDAR-LOC.
- The device receives data from LiDAR-LOC.

The communication protocols underneath the application layer (ISO-/OSI layers 1 ... 6) are not described.

It is assumed that the communication is transparent and error-free, with the following exceptions:

- Connection loss.
- Connection blockades when there are full transmit queues.

Device-specific deviations

- Byte sequence: LiDAR-LOC uses the Big Endian format for the data according to the bytes `Cmd` and `Mode`. Safety laser scanner, such as MICS3 or NANS3, use the Little Endian format.
  - Section: see "Layer 7.2, command layer", page 78)
- The TCP/IP for the CoLa2 communication of the safety laser scanner is port 2122
- A CoLa2 telegram can be split into fragments. The client must re-combine the fragments
- Methods of LiDAR-LOC can only be called up via name. Variable and methods for safety laser scanner, such as MICS3 or NANS3, can only be called up via index

5.1.1 Overview of the telegram format

The protocol defines 2 headers:

- Header of the layer 7.1, message layer

- Synchronization of the direct communication partners
- Specification of the length of the telegram

- Header of the layer 7.2, command layer
  - Session ID that connects 2 end-to-end partners
  - Request ID that connects a client request with the answer of the server (LiDAR-LOC)
  - Command that is sent to the server (LiDAR-LOC) or answer of the server
  - Additional information, depending on the command and the transmission direction

The session ID and request ID make it possible that several commands or answers can be sent to further communication between 2 partners without consideration. Several commands can also be transported in a TCP/IP packet.

With TCP/IP, all CoLa 2.0 connect requests must be sent to port 2122 of the server (LiDAR-LOC). With the TCP/IP socket of the client, you can configure TCP_NODELAY so that even small TCP/IP packages can be sent immediately and the end-to-end communication is accelerated.

5.1.1.1 Layer 7.1, message layer

The header of layer 7.1, message layer, ensures that the rest of the telegram is transmitted to the receiver that is described in the header.

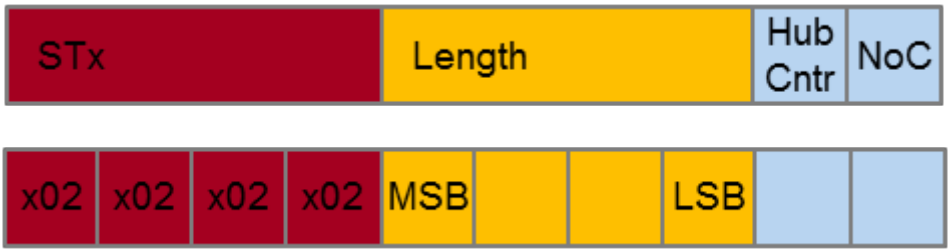

Figure 6: Header of the layer 7.1, message layer

Table 142: Header of the layer 7.1, message layer

Element	Length (bytes)	Function
STx	4	Start. 4 STx symbols (0x02) mark the start of each telegram. This pattern is not exclusive. Nevertheless, it allows you to find the start of a telegram after a synchronization loss in connection with the parameter `length`.
Length	4 (Big Endian)	Length. The number of bytes that follow as the rest of the telegram. After <length> bytes, 4 STx symbols should follow again, which mark the start of the next telegram.
HubCntr	1	
NoC	1	

5.1.1.2 Layer 7.2, command layer

The layer 7.2, command layer consists of a header and information.

Data flow takes place in 2 directions:

- From a client to server (LiDAR-LOC) for execution
- From a server (LiDAR-LOC) for evaluation

A complete transmission consists of a pair of these byte streams. After a method invocation (see "Calling up methods", page 81), the SICK sensor sends two answers: immediately a confirmation and later a result.

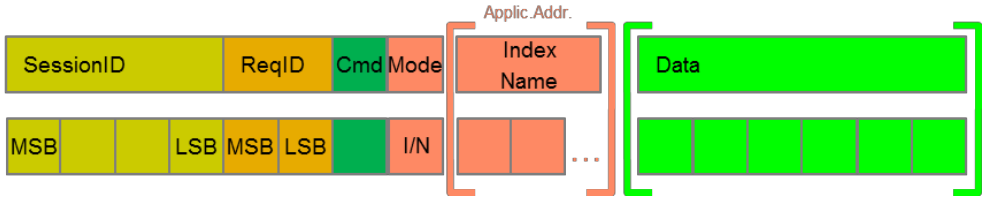

Figure 7: Layer 7.2, command layer

Table 143: Layer 7.2, command layer

Element	Length (bytes)	Function
SessionID	4 (Big Endian)	The server (LiDAR-LOC) defines this 32-bit value when structuring the session with a client. The entire communication between these two partners must be marked with the session ID.
ReqID	2 (Big Endian)	Using the ReqID, the client can assign the answer of the LiDAR-LOC its own request. Within one session, the client numbers all requests. The server (LiDAR-LOC) returns the ReqID together with its answer.
Cmd	1	This byte contains the command to the server (LiDAR-LOC). The server returns `Cmd` and changes the rest of the telegram according to the specification of the command.
Mode	1	Modifier for `Cmd`. All servers (SICK sensors, LiDAR-LOC) understand a basic command set. In addition, some variables are available for all servers for reading or writing. In addition, many SICK sensors have specific parameters.
IndexName, data	(Variable)	Specific data depending on `Cmd` and direction of transmission.

5.1.1.2.1 Byte sequence

The byte sequence after the bytes `Cmd` and `Mode` depends on the components of a sensor and differs for many different servers (SICK sensors):

- For Big Endian, the Most Significant Byte of a value is sent first (Motorola format).
- For Little Endian, the Least Significant Byte of a value is sent first (Intel format).

LiDAR-LOC uses Big Endian. Other sensors, e.g. the MICS3 and NANS3, use Little Endian.

The client must convert the values if needed.

Example: the variable `Angle` should be assigned the value 456 (0x1C8):

Byte sequence	Addressing	Request	Response
Big endian	Name	`WN_Angle_<0x01><0xC8>`	`WA_Angle_`
Little endian	Name	`WN_Angle_<0xC8><0x01>`	`WA_Angle_`

5.1.2 Sessions

The connection between a client (vehicle controller, computers etc.) and a server (LiDAR-LOC) is organized as a session. First a session must be established, only then can the partners communicate. Within a session, every communication exchange (each client request and accompanying server responses) is numbered with the ReqID. The server (LiDAR-LOC) creates the SessionID when setting up the session. The client creates a unique ReqID for each request to the server.

5.1.2.1 Setup of a session

Each client ((vehicle controller, computers etc.) must initiate at least one session, i.e. a direct connection to a server (LiDAR-LOC).

- For direct connection via Ethernet, you must open a socket for the IP address of LiDAR-LOC.

When setting up a session, the client can define several session parameters:

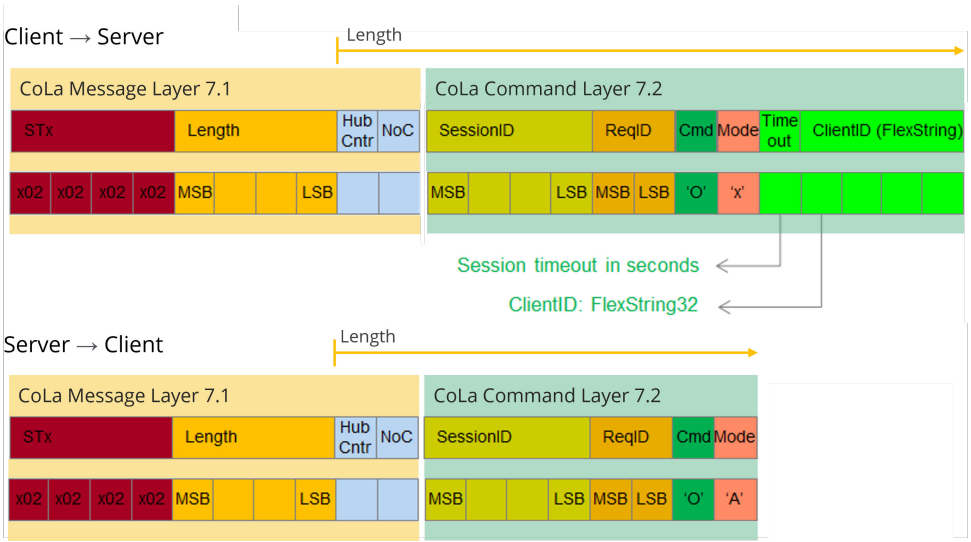

Figure 8: Setup of a session with a server (LiDAR-LOC)

The client sends a command to the server (LiDAR-LOC) via the selected hardware interface:

- HubCntr = 0
- NoC = 0
- SessionID = 0
- ReqID = unique value defined by client
- Cmd = 'O' (letter O, 0x4F), **O**pen session
- Mode = 'X' (letter X, 0x58), Other
- Timeout = number of seconds (binary, 1 ... 255)
- ClientID = identifier of the client (bytestream)

The server (LiDAR-LOC) returns the command with the following changes:

- SessionID = The server sends the session ID that the client must use for all further requests to the server.
- Cmd = 'O' (letter O, 0x4F), **O**pen session
- Mode = 'A' (letter A, 0x41), Application **A**nswer
- The Timeout and ClientID fields are truncated.

The set-up session exists up to explicit completion or up to the timeout.

If the server (LiDAR-LOC) does not receive a command from the client within `Timeout` seconds, it ends the session. Then the client answers requests with the `SessionID` with the error message "Invalid Session".

The client must send requests as often as necessary so that the timeout does not expire. You can implement a timer on the client for this that is reset for each request. After the timer has run out, the client should send a dummy command to maintain the session.

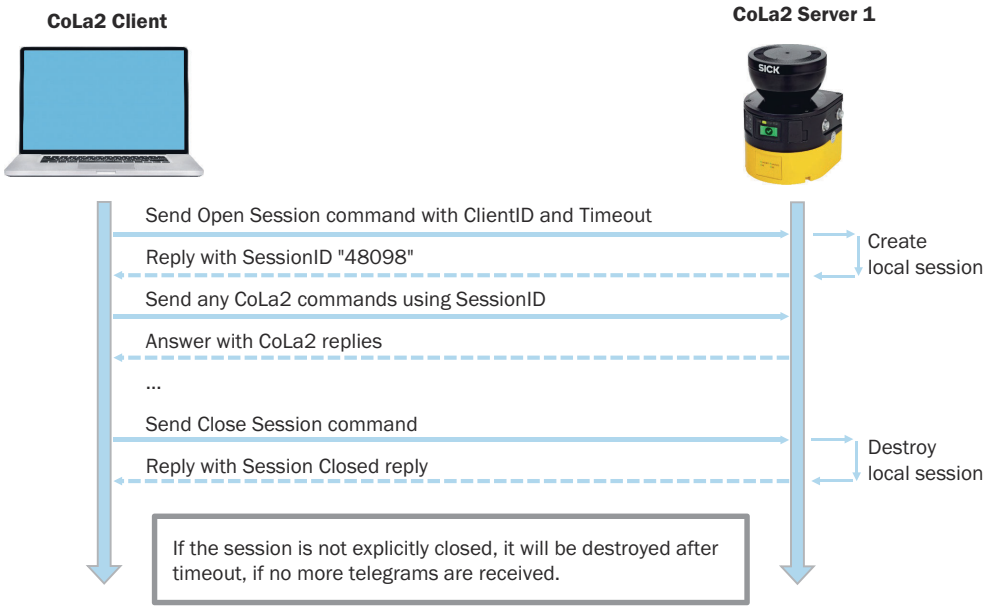

Figure 9: Expiration of a session

Command to end a session:

- Cmd = 'C' (letter C, 0x43), **C**lose session
- Mode = 'X' (letter X, 0x58), Other

Server response:

- Cmd = 'C' (letter C, 0x43), **C**lose session
- Mode = 'A' (letter A, 0x41), Application **A**nswer

5.1.3 Using LiDAR-LOC with CoLa2

A client (vehicle controller, computer etc.) running LiDAR-LOC uses only the followin command type in operation:

- Call up methods, i.e. activate a routine in LiDAR-LOC.

The following describes how the commands need to be structured and how the responses of the LiDAR-LOC can be evaluated.

The variables and methods are addressed using their names. LiDAR-LOC always responds with the names.

5.1.3.1 Calling up methods

A method is a program in LiDAR-LOC that the parameters are transmitted to and that can be started.

The client sends a command to the server (LiDAR-LOC) via the selected hardware interface:

- Cmd = 'M' (letter M, 0x4D), Invoke **M**ethod
- Mode = 'N' (letter N, 0x4E), By **N**ame

The server (LiDAR-LOC) returns the command:

- Cmd = 'A' (letter A, 0x41), Method **A**nswer
- Mode = 'N' (letter N, 0x4E), By **N**ame

5.1.3.2 Error messages

The server (LiDAR-LOC) sends error numbers in the following format:

Index	Cmd = F	Mode = A	Data = (UINT)

- Cmd = ‘F’ (letter F, 0x46), **F**ailure
- Mode = ‘A’ (letter A, 0x41), Application **A**nswer

The error number is sent in the sensor-specific byte sequence, see "Byte sequence", page 79. SessionID and ReqID identify the command that caused the error.

Table 144: Fault numbers

Fault number	Name	Description
0x0001	METHODIN_ACCESSDENIED	Incorrect user group. Calling up the method not allowed.
0x0002	METHODIN_UNKNOWNINDEX	Method with the specified SOPAS index is not known.
0x0003	VARIABLE_UNKNOWNINDEX	Variable with the specified SOPAS index is not known.
0x0004	LOCALCONDITIONFAILED	Local condition infringed. Example: Specified value is outside the permissible range for the variable.
0x0005	INVALID_DATA	Invalid data for variable.
0x0006	UNKNOWN_ERROR	Errors with unknown cause.
0x0007	BUFFER_OVERFLOW	Communication buffer too small for the data amount to be serialized.
0x0008	BUFFER_UNDERFLOW	More data was expected. The assigned buffer could not be filled.
0x0009	ERROR_UNKNOWN_TYPE	The variable has an unknown type. There are undocumented internal variables in the firmware of the device.
0x000A	VARIABLE_WRITE_ ACCESS_DENIED	No values could be written in this variable.
0x000B	UNKNOWN_CMD_ FOR_NAME-SERVER	When calling up via the name: Name of the command is not known to the name server.
0x000C	UNKNOWN_COLA_COMMAND	Name of the command is not defined in the CoLa protocol. Name of the command is not known.
0x000D	METHODIN_SERVER_BUSY	Only one method can be sent to the device at the same time.
0x000E	FLEX_OUT_OF_BOUNDS	Array has the incorrect length.
0x000F	EVENTREG_UNKNOWNINDEX	When calling up via index: Event is not known.
0x0010	COLA_A_VALUE_OVERFLOW	Value is too large for the value field.
0x0011	COLA_A_INVALID_CHARACTER	Symbol unknown, likely not alphanumeric.
0x0012	OSAI_NO_MESSAGE	General error when reading a variable.
0x0013	OSAI_NO_ANSWER_MESSAGE	General error when writing to a variable.
0x0014	INTERNAL	Internal firmware error

Fault number	Name	Description
0x0015 ... 0x0018	n/a	Reserved.
0x0019	AsyncMethodsAreSuppressed	An asynchronous method was called up, but the device does not permit any asynchronous methods.
0x001A ... 0x001F	n/a	Reserved.
0x0020	ComplexArraysNotSupported	A complex array was found, but the device does not permit any complex arrays.
0x0021	SESSION_NORESOURCES	Session cannot be structured. Server resources are exhausted as too many clients are connected.
0x0022	SESSION_UNKNOWNID	Unknown session ID in the telegram header
0x0023	CANNOT_CONNECT	Connection cannot be established.
0x0024	InvalidPortId	PortID is not known to the server.
0x0025 ... 0x0029	n/a	Reserved.
0x0030	-	Out of range.
0x0031	-	Invalid identifier .
0x0032	-	Not found .
0x0033	-	Parsing failed.
0x0034	-	A structured INode is invalid.
0x0035	-	No method handler set.
0x0036	-	Name not set.
0x0037	-	Invalid state.
0x0038	-	File is in bad state.
0x0039	-	No challenge.
0x0040	-	Password not changeable.
0x0041	-	Login error, might be due to timelock or invalid user/ password.
0x0042	-	Unexpected response (client).
0x0043	-	Thread communication is broken.
0x0044	-	No index set (client).
0x0045	-	Invalid challenge (client).
0x0046	-	Error during CID parsing, CID is broken.
0x0047	-	Default value parsing out of CID found an error.
0x0048	-	Invalid or unsupported UTF-8 character.
0x0049	-	SOPAS file format version is not supported.
0x0050	-	SOPAS file is broken or cannot be handled.

Fault number	Name	Description
0x0051	-	Parsing a name from a cloning string.
0x0052	-	Send UDP data failed.
0x0053	-	XML error.

5.1.4 CoLa2 data types

Table 145: CoLa2 data types

Name	Description	Value range
BOOL	Boolean	True (1), false (0)
USINT	Unsigned short (8 bit)	0 ... 255
UINT	Unsigned int (16 bit)	0 ... 65535
UDINT	Unsigned double int (32 bit)	0 ... 4294967295
ULINT	Unsigned long int (64 bit)	0 ... 18446744073709551616
SINT	Signed short (8 bit)	-128 ... 127
INT	Signed int (16 bit)	-32768 ... 32767
DINT	Signed double int (32 bit)	-2147483648 ... 2147483647
LINT	Signed long int (64 bit)	-9223372036854775808 ... 9223372036854775807
REAL	Float single precision (32 bit)	See IEEE 754
LREAL	Float double precision (64 bit)	See IEEE 754
ENUM8	Short enumeration (8 bit)	Values defined in a selection list (0 ... 255)
ENUM16	Enumeration (16 bit)	Values defined in a selection list (0 ... 65535)
STRING	Array of visible symbols (array of 8-bit symbols)	Symbol = USInt with a value between 0x20 ... 0xFF [1)]
FLEXSTRING	Array of visible symbols with leading length specification (UInt) (array of 8-bit symbols)	See string and FlexArray
BYTE	Bitset (8 bit)	Value is transmitted as an array from USInt.
WORD	Bitset (16 bit)	Value is transmitted as an array from USInt.
DWORD	Bitset (32 bit)	Value is transmitted as an array from USInt.
LWORD	Bitset (64 bit)	Value is transmitted as an array from USInt.
XBYTE	Bitset (8, 16, 24, 32 ... bit)	Value is transmitted as an array from USInt.
SCONT	Bitset (8 bit)	Value is transmitted as USInt.
CONT	Bitset (16 bit)	Value is transmitted as UInt.
DCONT	Bitset (32 bit)	Value is transmitted as UDInt.
LCONT	Bitset (64 bit)	Value is transmitted as ULInt.
STRUCTURE	Sequence of various types	Possible types: basic types, structures, arrays.
ARRAY	Repetition of a type	The length is defined for every array. Possible types: basic types, structures, arrays.

Name	Description	Value range
FLEXARRAY	Repetition of variable length of one type	The maximum length is defined for every FlexArray. The actual length is transmitted as UInt at the start of the FlexArray. Possible types: basic types, structures, arrays.

1) Coded according to ISO 8859-15.

5.2 Appendix D: Examples of communication via CoLa2

This chapter describes how the connection to LiDAR-LOC is established via the CoLa2 network (TCP/IP) and the method is called-up.

It is strongly recommended to familiarize yourself with CoLa2 communication in advance, see "Appendix A: Communication via CoLa2", page 77.

Table 146: CoLa2 elements

Element	sTX	Length	Hub	NoC	SessionID[1)]	ReqID	CMD	Mode	TimeOut[2)]	ClientID (FlexString)	
										Length[3)]	String
Length (bytes)	4	4	1	1	4	2	1	1	1	2	flex

1) The server (LiDAR-LOC) creates a unique SessionID when setting up the session.
2) Maximum timeout is 127 sec, 0x7F. Timeout is specified only when opening a session.
3) The length of the flexstring is used for only for methods with parameter type "string".

In the example, the CoLa2 and the TCP sessions are closed after calling the methods. Otherwise LiDAR-LOC would close the TCP session after the configured timeout of 30 seconds (0x1E). Data output via UDP is independent from a CoLa2 session and also continues to run after the end of a session. For configuration changes a new CoLa2 session can be established.

5.2.1 Example 1 - CoLa2: Setting a map

This chapter describes how the `LocSetMap` method is called up to activate a given map.

1. Open TCP session to LiDAR-LOC, port 2122.
2. Open CoLa2 session. To do so, send a CoLa2 telegram to establish a session (STX = 0x02020202, length = 14_{dec}, 0xE, Hub = 0, NoC = 0, SessionID = 0, ReqID = 1, Cmd = “O”, Mode = “X”, Timeout=30sec, ClientID Length = 1, ClientID Flexstring = 0)

```
--> 02020202 0000000E 00 00 00000000 0001 4F 58 1E 0001 00
```

✓ LiDAR-LOC confirms the command (length = 10_{dec}, 0xA, Cmd = “O”, Mode = “A”) and assigns a session ID (SessionID).

```
<-- 02020202 0000000A 00 00 2D6C2733 0001 4F 41
```

3. Call up method **LocSetMap** (length = 33_{dec}, 0x21, Cmd = “M”, Mode = “N”, Space (0x20) between method and parameter, ClientID Length = 11_{dec}, 0xB, ClientID Flexstring (parameter) = “mytest.vmap”)

```
--> 02020202 00000021 00 00 2D6C2733 0002 4D 4E
4C6F635365744D6170 20 000B 6D79746573742E766D6170
```

✓ LiDAR-LOC confirms the method call-up (length = 22_{dec}, 0x16, Cmd = “A”, Mode = “N”, FlexString success result = "1").

```
<-- 02020202 00000016 00 00 2D6C2733 0002 41 4E 20
4C6F635365744D6170 20 01
```

4. Close CoLa2 session (length = 10_{dec}, 0xA, Cmd = “C”, Mode = “X”).

```
--> 02020202 0000000A 00 00 2D6C2733 0003 43 58
```

✓ LiDAR-LOC confirms the command (Cmd = “C”, Mode = “A”).

```
<-- 02020202 0000000A 00 00 2D6C2733 0003 43 41
```

5. Close the TCP session.

5.2.2 Example 2 - CoLa2: Initialize at pose

This chapter describes how the `LocInitializeAtPose` method is called-up to initialize at a given pose.

1. Open TCP session to LiDAR-LOC, port 2122.
2. Open CoLa2 session. To do so, send a CoLa2 telegram to establish a session (STX = 0x02020202, length = 14_{dec}, 0xE, Hub = 0, NoC = 0, SessionID = 0, ReqID = 1, Cmd = “O”, Mode = “X”, Timeout=30sec, ClientID Length = 1, ClientID Flexstring = 0)

```
--> 02020202 0000000E 00 00 00000000 0001 4F 58 1E 0001 00
```

✓ LiDAR-LOC confirms the command (length = 10_{dec}, 0xA, Cmd = “O”, Mode = “A”) and assigns a session ID (SessionID).

```
<-- 02020202 0000000A 00 00 2D6C2733 0001 4F 41
```

3. Call up method **LocInitializeAtPose** (length = 44_{dec}, 0x2C, Cmd = “M”, Mode = “N”, Space (0x20) between method and parameter, ClientID Flexstring: parameter 1 = X coordinate: 10.3 m, 10 300 mm, parameter 2 = Y coordinate: -5.2 m, -5 200 mm, parameter 3 = Yaw angle: 30 deg, 30 000mdeg, parameter 4 = searchRadius: 1 m, 1 000 mm

```
--> 02020202 0000002C 00 00 2D6C2733 0002 4D 4E
4C6F63496E697469616C697A654174506F7365 20 00000406 FFFFEBB0
00007530 03E8
```

✓ LiDAR-LOC confirms the method call-up (length = 32_{dec}, 0x20, Cmd = “A”, Mode = “N”, FlexString success result = "1").

```
<-- 02020202 00000020 00 00 2D6C2733 0002 41 4E 20
4C6F63496E697469616C697A654174506F7365 20 01
```

4. Close CoLa2 session (length = 10_{dec}, 0xA, Cmd = “C”, Mode = “X”).

```
--> 02020202 0000000A 00 00 2D6C2733 0003 43 58
```

✓ LiDAR-LOC confirms the command (Cmd = “C”, Mode = “A”).

```
<-- 02020202 0000000A 00 00 2D6C2733 0003 43 41
```

5. Close the TCP session.

8027127/1NIC/2024-05-17/en

Australia
Phone +61 (3) 9457 0600
1800 33 48 02 – tollfree
E-Mail sales@sick.com.au

Austria
Phone +43 (0) 2236 62288-0
E-Mail office@sick.at

Belgium/Luxembourg
Phone +32 (0) 2 466 55 66
E-Mail info@sick.be

Brazil
Phone +55 11 3215-4900
E-Mail comercial@sick.com.br

Canada
Phone +1 905.771.1444
E-Mail cs.canada@sick.com

Czech Republic
Phone +420 234 719 500
E-Mail sick@sick.cz

Chile
Phone +56 (2) 2274 7430
E-Mail chile@sick.com

China
Phone +86 20 2882 3600
E-Mail info.china@sick.net.cn

Denmark
Phone +45 45 82 64 00
E-Mail sick@sick.dk

Finland
Phone +358-9-25 15 800
E-Mail sick@sick.fi

France
Phone +33 1 64 62 35 00
E-Mail info@sick.fr

Germany
Phone +49 (0) 2 11 53 010
E-Mail info@sick.de

Greece
Phone +30 210 6825100
E-Mail office@sick.com.gr

Hong Kong
Phone +852 2153 6300
E-Mail ghk@sick.com.hk

Hungary
Phone +36 1 371 2680
E-Mail ertekesites@sick.hu

India
Phone +91-22-6119 8900
E-Mail info@sick-india.com

Israel
Phone +972 97110 11
E-Mail info@sick-sensors.com

Italy
Phone +39 02 27 43 41
E-Mail info@sick.it

Japan
Phone +81 3 5309 2112
E-Mail support@sick.jp

Malaysia
Phone +603-8080 7425
E-Mail enquiry.my@sick.com

Mexico
Phone +52 (472) 748 9451
E-Mail mexico@sick.com

Netherlands
Phone +31 (0) 30 204 40 00
E-Mail info@sick.nl

New Zealand
Phone +64 9 415 0459
0800 222 278 – tollfree
E-Mail sales@sick.co.nz

Norway
Phone +47 67 81 50 00
E-Mail sick@sick.no

Poland
Phone +48 22 539 41 00
E-Mail info@sick.pl

Romania
Phone +40 356-17 11 20
E-Mail office@sick.ro

Singapore
Phone +65 6744 3732
E-Mail sales.gsg@sick.com

Slovakia
Phone +421 482 901 201
E-Mail mail@sick-sk.sk

Slovenia
Phone +386 591 78849
E-Mail office@sick.si

South Africa
Phone +27 10 060 0550
E-Mail info@sickautomation.co.za

South Korea
Phone +82 2 786 6321/4
E-Mail infokorea@sick.com

Spain
Phone +34 93 480 31 00
E-Mail info@sick.es

Sweden
Phone +46 10 110 10 00
E-Mail info@sick.se

Switzerland
Phone +41 41 619 29 39
E-Mail contact@sick.ch

Taiwan
Phone +886-2-2375-6288
E-Mail sales@sick.com.tw

Thailand
Phone +66 2 645 0009
E-Mail marcom.th@sick.com

Turkey
Phone +90 (216) 528 50 00
E-Mail info@sick.com.tr

United Arab Emirates
Phone +971 (0) 4 88 65 878
E-Mail contact@sick.ae

United Kingdom
Phone +44 (0)17278 31121
E-Mail info@sick.co.uk

USA
Phone +1 800.325.7425
E-Mail info@sick.com

Vietnam
Phone +65 6744 3732
E-Mail sales.gsg@sick.com

Detailed addresses and further locations at **www.sick.com**

SICK AG | Waldkirch | Germany | www.sick.com

SICK
Sensor Intelligence.