rexroth
A Bosch Company

Work in Progress: Laser Localization Software – API Documentation for Version 1.2

Contents

1 Preface **10**
- 1.1 General Notes . . . 10
- 1.2 About this document . . . 11
- 1.3 Structure . . . 11
- 1.4 Naming Conventions . . . 11

2 API Modules **12**
- 2.1 Version Numbering . . . 13
- 2.2 API Module Overview . . . 13

3 Interfaces **14**
- 3.1 RPC Interface . . . 15
  - 3.1.1 Call . . . 15
  - 3.1.2 Response . . . 15
  - 3.1.3 Error . . . 16
  - 3.1.4 Message Naming Conventions . . . 17
  - 3.1.5 Session IDs . . . 17
  - 3.1.6 Remarks on the HTTP Server . . . 17
- 3.2 Binary Interface Concept . . . 18
  - 3.2.1 Push Provider . . . 18
  - 3.2.2 Push Consumer . . . 19
  - 3.2.3 Datagram Naming Conventions . . . 19

4 Communication Security **19**
- 4.1 Concept . . . 19
- 4.2 Requirements . . . 20
- 4.3 Certificate Management . . . 20

5 Client Control Mode **21**

6 Response Codes **22**
- 6.1 Preface . . . 22
- 6.2 Module identifiers . . . 22
- 6.3 Common Response Codes . . . 23
- 6.4 *Session*-Specific Response Codes . . . 25
- 6.5 *LicensingFeature*-Specific Response Codes . . . 25
- 6.6 *Config*-Specific Response Codes . . . 27
- 6.7 *Certificate*-Specific Response Codes . . . 28
- 6.8 *ClientMap*-Specific Response Codes . . . 28

6.9 *ClientGlobalAlign*-Specific Response Codes 29
6.10 *SupportRecovery*-Specific Response Codes 29

7 Common JSON Objects and Types **30**
7.1 Types 30
7.1.1 AsciiCharacterString (string) 30
7.1.2 EmptyString (string) 30
7.1.3 Integer64 (integer) 30
7.1.4 NonnegativeInteger64 (integer) 31
7.1.5 IEEE754Double (number) 31
7.1.6 ResponseCode (NonnegativeInteger64) 31
7.1.7 SessionId (string) 31
7.1.8 Size (NonnegativeInteger64) 31
7.1.9 RecordingName (AsciiCharacterString, length > 0) 31
7.1.10 ClientMapName (AsciiCharacterString, length > 0) 32
7.1.11 ServerMapName (AsciiCharacterString, length > 0) 32
7.1.12 NetworkAddress(AsciiCharacterString, length > 0) 32
7.2 Objects 32
7.2.1 Timestamp 32
7.2.2 TimeInterval 32
7.2.3 Pose3D 33
7.2.4 Transform3D 33
7.2.5 Pose2D 33
7.2.6 Transform2D 34
7.2.7 Container 34

8 Shared (Localization Client / Map Server / Support) – RPC Methods **35**
8.1 API Module Information (Module: *AboutModules*) 35
8.1.1 aboutModulesList 35
8.1.2 Messages 35
8.1.3 Objects 36
8.2 Session Control (Module: *Session*) 36
8.2.1 sessionLogin 36
8.2.2 sessionLogout 37
8.2.3 sessionRefresh 37
8.2.4 sessionGroupInfo 37
8.2.5 Messages 38
8.3 Diagnostic Information (Module: *Diagnostic*) 39
8.3.1 diagnosticList 39
8.3.2 diagnosticClear 39

8.3.3 Messages . . . 39
8.3.4 Objects . . . 40
8.4 Software Licensing (Module: *LicensingFeature*) . . . 41
8.4.1 licensingFeatureGetTrustedPlatformModuleInformation . . . 41
8.4.2 licensingFeatureGetHostId . . . 42
8.4.3 licensingFeatureSet . . . 42
8.4.4 licensingFeatureList . . . 42
8.4.5 Messages . . . 43
8.4.6 Objects . . . 44
8.5 System Configuration (Module: *Config*) . . . 45
8.5.1 configList . . . 45
8.5.2 configSet . . . 46
8.5.3 Messages . . . 46
8.5.4 Objects . . . 47
8.6 System Information (Module: *AboutBuild*) . . . 52
8.6.1 aboutBuildList . . . 52
8.6.2 Messages . . . 53
8.7 Certificate Management (Module: *Certificates*) . . . 53
8.7.1 certificateSet . . . 53
8.7.2 Messages . . . 54
8.8 System Control (Module: *System*) . . . 54
8.8.1 systemShutdown . . . 54
8.8.2 Messages . . . 54

9 Localization Client – RPC Methods 55
9.1 Recording (Module: *ClientRecording*) . . . 55
9.1.1 clientRecordingStart . . . 55
9.1.2 clientRecordingStop . . . 55
9.1.3 clientRecordingStartVisualRecording . . . 56
9.1.4 clientRecordingStopVisualRecording . . . 56
9.1.5 clientRecordingList . . . 57
9.1.6 clientRecordingRename . . . 57
9.1.7 clientRecordingDelete . . . 57
9.1.8 clientRecordingSetCurrentPose . . . 58
9.1.9 Messages . . . 58
9.2 Client-Side Map Management (Module: *ClientMap*) . . . 59
9.2.1 clientMapStart . . . 59
9.2.2 clientMapStop . . . 60
9.2.3 clientMapList . . . 60
9.2.4 clientMapRename . . . 60

9.2.5 clientMapDelete . . . 61
9.2.6 clientMapSend . . . 61
9.2.7 Messages . . . 62
9.3 Client Localization (Module: *ClientLocalization*) . . . 63
9.3.1 clientLocalizationStart . . . 63
9.3.2 clientLocalizationStop . . . 63
9.3.3 clientLocalizationSetSeed . . . 64
9.3.4 Messages . . . 64
9.4 Manual Map Alignment (Module: *ClientManualAlign*) . . . 65
9.4.1 clientManualAlignStart . . . 65
9.4.2 clientManualAlignStop . . . 65
9.4.3 clientManualAlignGetPointCloud . . . 65
9.4.4 clientManualAlignSet . . . 66
9.4.5 Messages . . . 67
9.4.6 Objects . . . 68
9.4.7 ClientManualAlignOriginTransform2D . . . 68
9.5 Global Alignment (Module: *ClientGlobalAlign*) . . . 68
9.5.1 clientGlobalAlignAddObservation . . . 68
9.5.2 clientGlobalAlignReplaceObservations . . . 68
9.5.3 clientGlobalAlignDeleteObservations . . . 69
9.5.4 clientGlobalAlignListObservations . . . 70
9.5.5 Messages . . . 70
9.5.6 Objects . . . 71
9.5.7 ClientGlobalAlignSensorTransform2D . . . 72
9.5.8 ClientGlobalAlignLandmarkPose2D . . . 72
9.5.9 Types . . . 73
9.6 Laser Masking (Module: *ClientLaserMask*) . . . 74
9.6.1 clientLaserMaskingStart . . . 74
9.6.2 clientLaserMaskingStop . . . 74
9.6.3 clientLaserMaskingGetScan . . . 75
9.6.4 Messages . . . 75
9.7 User Account Management (Module: *ClientUser*) . . . 76
9.7.1 Types . . . 76
9.7.2 clientUserAdd . . . 76
9.7.3 clientUserDelete . . . 77
9.7.4 clientUserChangePassword . . . 77
9.7.5 clientUserChangeOwnPassword . . . 78
9.7.6 clientUserList . . . 78
9.7.7 clientUserListGroup . . . 79
9.7.8 Messages . . . 79

9.7.9 Objects 80

10 Map Server – RPC Methods **81**
10.1 Server-side Map Management (Module: *ServerMap*) 81
10.1.1 serverMapGetInfo 81
10.1.2 serverMapList 82
10.1.3 serverMapRename 82
10.1.4 serverMapDelete 83
10.1.5 serverMapGetPointCloud 83
10.1.6 serverMapGetImage 84
10.1.7 serverMapGetImageWithResolution 84
10.1.8 serverMapGetMapThumbnail 85
10.1.9 Messages 86
10.2 Server-side Internal Settings (Module: *ServerInternal*) 87
10.2.1 serverInternalFleetConfigSet 88
10.2.2 serverInternalFleetConfigGet 88
10.2.3 Messages 88
10.2.4 Objects 89
10.3 User Management (Module: *ServerUser*) 89
10.3.1 Types 89
10.3.2 serverUserAdd 89
10.3.3 serverUserDelete 90
10.3.4 serverUserChangePassword 90
10.3.5 serverUserChangeOwnPassword 91
10.3.6 serverUserList 91
10.3.7 serverUserListGroup 92
10.3.8 Messages 92
10.3.9 Objects 93

11 Support – RPC Methods **94**
11.1 Support Reports (Module: *SupportReport*) 94
11.1.1 supportReportCreate 94
11.1.2 supportReportCreateMinimal 94
11.1.3 supportReportSetDescription 95
11.1.4 supportReportList 95
11.1.5 supportReportGetPath 96
11.1.6 supportReportDelete 96
11.1.7 Messages 97
11.1.8 Objects 98
11.2 System Backup, Update, Recovery (Module: *SupportRecovery*) 98

11.2.1 supportRecoveryList . . . 98
11.2.2 supportRecoveryCreate . . . 98
11.2.3 supportRecoveryDelete . . . 99
11.2.4 supportRecoveryFactoryReset . . . 99
11.2.5 supportRecoveryFrom . . . 100
11.2.6 Messages . . . 100
11.2.7 Objects . . . 101

12 Binary Interfaces **101**
12.1 Common Definitions (Shared across Modules) . . . 101
12.1.1 Common Types . . . 101
12.1.2 Common Constants . . . 103
12.1.3 Arrays . . . 103
12.1.4 Common Datagrams . . . 104
12.2 Control Output (Module: *ClientControl*) . . . 104
12.2.1 Interfaces . . . 104
12.2.2 Datagrams . . . 105
12.3 Localization Output (Module: *ClientLocalization*) . . . 106
12.3.1 Interfaces . . . 106
12.3.2 Datagrams . . . 106
12.4 Visual Recording Output (Module: *ClientRecording*) . . . 109
12.4.1 Interfaces . . . 109
12.4.2 Datagrams . . . 110
12.5 Mapping Output (Module: *ClientMap*) . . . 112
12.5.1 Interfaces . . . 112
12.5.2 Datagrams . . . 112
12.6 Global Alignment (Module: *ClientGlobalAlign*) . . . 114
12.6.1 Interfaces . . . 114
12.6.2 Datagrams . . . 114
12.7 Sensor Input (Module: *ClientSensor*) . . . 116
12.7.1 Interfaces . . . 116
12.7.2 Datagrams . . . 118
12.8 FAQ . . . 123

13 User Groups **124**

14 Licensing **127**
14.1 Methods for host machine identification . . . 127
14.2 Licensing for API methods . . . 127
14.3 Licensing and the Localization Client . . . 129
14.3.1 Licensing and specific Client features . . . 130

14.4 Licensing and the Map Server . . . 130
14.5 Licensing and the Support Component . . . 130

15 Appendix **130**
15.1 Code Examples . . . 130
15.1.1 Binary Interface . . . 130

16 Changelog **131**
16.1 Version 1.2 . . . 131
16.1.1 Clarifications regarding user interfaces . . . 131
16.1.2 Changes to the ClientGlobalAlign module . . . 131
16.1.3 Changes to the ClientLocalization module . . . 132
16.1.4 Changes to the ClientManualAlign module . . . 132
16.1.5 Changes to the ClientSensor module . . . 132
16.1.6 Changes to the ClientUser module . . . 133
16.1.7 Changes to the Config module . . . 133
16.1.8 Changes to the LicensingFeature module . . . 133
16.1.9 Changes to the Diagnostic module . . . 134
16.1.10 Changes to the ServerMap module . . . 134
16.1.11 New module: ServerInternal . . . 134
16.1.12 Changes to the ServerUser module . . . 134
16.2 Version 1.1 . . . 134
16.2.1 Network ports for Client-Server communication . . . 134
16.2.2 Clarifications regarding licensing . . . 135
16.2.3 Response Codes . . . 135
16.2.4 Protecting entities from inadvertent deletion . . . 135
16.2.5 Protecting entities in use . . . 135
16.2.6 Ensuring coordinate precision . . . 136
16.2.7 Handling irrational numbers . . . 136
16.2.8 Visualization Ids . . . 136
16.2.9 Changes to the LicensingFeature module . . . 136
16.2.10 Changes to the Config module . . . 137
16.2.11 Changes to the Certificate module . . . 137
16.2.12 Changes to the ClientRecording module . . . 137
16.2.13 Changes to the ClientMap module . . . 137
16.2.14 Changes to the ClientManualAlign module . . . 138
16.2.15 Changes to the ClientGlobalAlign module . . . 138
16.2.16 Changes to the ClientUser module . . . 138
16.2.17 Changes to the ServerMap module . . . 138
16.2.18 Changes to the ServerUser module . . . 138

16.2.19 Changes to the SupportReport module 138
16.2.20 Changes to the SupportRecovery module 139
16.3 Version 1.0 139

1 Preface

This is the interface documentation for the Bosch Rexroth Laser Localization Software (LLS) Version 1.2. Copyright 2018-2020 Bosch Rexroth.

1. API Modules: an introduction to the concept of API modularization, API module versions and a list of current API modules.
2. Interfaces: an introduction to the two main interfaces provided by the Laser Localization Software, including their respective mechanics and nomenclatures.
3. Communication Security explains all concepts serving security such as encryption and certificate management.
4. Client Control Mode lists the different modes the Localization Client can be set to run in, e.g. mapping or localization.
5. Response Codes defines the response codes which are always part of a response to a query sent to the RPC Interface.
6. Common JSON Objects and Types defines JSON Objects and Types which are the data building blocks of RPC Interface queries and responses.
7. Shared (Localization Client / Map Server / Support) – RPC Methods lists methods of the RPC Interface which are provided by all elements of the Laser Localization Software (Map Server, Localization Client, Support).
8. Localization Client – RPC Methods lists methods of the Localization Client's RPC Interface.
9. Map Server – RPC Methods lists methods of the Map Server's RPC Interface.
10. Support – RPC Methods lists methods of the Support Component's RPC Interface.
11. Binary Interfaces defines the Binary Interfaces provided by the Laser Localization Software based on the introduction in the Chapter on Interfaces.
12. User Groups explains the different groups involved in user account management and authentication.
13. Appendix contains an overview of the code examples included with the document.

1.1 General Notes

Bosch Rexroth reserves the right to change (technical) features and characteristics of existing API interfaces with new releases (updates or upgrades, workarounds) of the Software. Thus, Bosch Rexroth does not assume any warranty or liability with regard to API interfaces. The customer is aware that after a release of the Software, the API interface counterpart may need to be updated as well.

1.2 About this document

This document provides the precise technical specifications for the RPC and Binary Interfaces used by the Bosch Rexroth Laser Localization Software. It supplements the Laser Localization Software manual, which describes the overall system, its capabilities, and how to use it. In contrast, this interface documentation specifies how exactly the Laser Localization Software's interfaces should be used. Consequently, the user should first study the software manual to understand the system. They should then refer to this document when actually writing software that interacts with the Laser Localization Software. All interfaces described here are strictly modularized and versioned. The user should therefore ensure that the document version matches the API version, as per the appropriate chapter.

1.3 Structure

This document is split into several segments, as listed in the table of contents: First, it introduces the user to core concept such as API modules, the RPC and Binary Interfaces, and how to securely communicate with those interfaces.

Next, the document covers the RPC interface, which is mainly used for controlling the Laser Localization Software. This includes general aspects, such as the use of Response Codes and the common objects and types used throughout the interface. At its core, the document lists all RPC methods available on the various product elements. This includes the Localization Client and the Map Server, as well as those methods shared between them.

The following major section discusses the Binary Interfaces, which handle low-latency and high-throughput data transfer.

The document concludes with an appendix that lists the code examples included with this document. These examples demonstrate how to operate some of the interfaces described here.

1.4 Naming Conventions

- A Laser Localization System consists of a Map Server, at least one Localization Client, and one or more optional Support Components. For better readability in source code or API these elements will sometimes be referred to as *Server*, *Client*, and *Support Component* (or abbreviated *Support*) within this document.

- Types, methods, variables, module names will in camel case style, types and module names starting with a capital letter, methods and variables not (e.g., DiagnosticArrayMessage or sessionId).
- Abbreviations start with a capital letter, followed by lower case letters (e.g., Id)
- An exception is the style of coordinate systems, which will be always in upper case letters (e.g., MAP, LASER) because of their special and error-prone meaning. Variables whose inputs are relative to a coordinate system, gets the name of the system as prefix (e.g. MAPpose2D). Transformation also uses an affix for the source system (e.g., VEHICLEstaticCalibLASER describing the static calibration from the LASER system into the VEHICLE system).
- Timestamps used in text-based log messages conform to the ISO 8601 basic format. The map and recording name is specified by the customer user. But it is recommended to include a timestamp in ISO 8601 basic format 'YYYYMMDDTHHMMSSZ' in UTC, e.g. 20200330T123015Z in the file name to make the file distinguishable.

2 API Modules

The interface of the Laser Localization Software is split in different API modules. An API module is a collection of RPC API methods and messages for a specific task (e.g., user account management, mapping etc.). Each module may also encompass one or more Binary Interfaces and their associated data types. All API modules are independent of each other and contain their own specific name and version number. Users can check the API modules included in their software through the aboutModulesList method. This method returns a list of all supported API module names and their version numbers. Changes made to one of the modules will not affect the others, which keeps the API easy to use even as it evolves. User-made programs that interact with the Laser Localization Software should always check the actual API module versions. This ensures that the program is compatible with this specific instance of the Laser Localization Software.

The name of an API module is a string which will remain unchanged throughout all future LLS releases. If the task of the API module is available for both the Localization Client and the Map Server the name has no prefix (e.g. *Session*); otherwise the prefix (Server, Client or Support) signals which element the API module belongs to (e.g. *ClientLaserMask* or *ServerMap*).

The modules implemented by the Support Component bear the prefix *Support*. Note that Support Components only contain a reduced subset of the common modules.

Besides the SupportReport and SupportRecovery modules exclusive to the Support Component, it only implements the modules AboutModules and Session.

2.1 Version Numbering

The API module version number is defined as: *MajorNumber.MinorNumber*

The *MajorNumber* increases when at least one API method, message, data type, Binary Interface, or Datagram is no longer backwards compatible with respect to the previous versions of the API module. Consequently, programs written to work with such a previous version might not work with the new version of the module. * The *MinorNumber* increases when a module changes in a manner that is backwards compatible. For example, a new method may be added to the module which implements some novel functionality. Programs written to work with a different module version will still work if only the minor version increases. Note that such a program may not work with a lower minor version, as some new functionality may not be available.

2.2 API Module Overview

The following list shows all available API Modules, their names, versions and short descriptions. Refer to the changelog for information on how these modules changed with each software release.

Name of API Module	Version	Description
AboutModules	4.0	get names and versions of all API-Modules which are supported from the current product version
Session	3.0	manage access to the system
Diagnostic	3.0	get diagnostic information
LicensingFeature	4.0	handle licenses and get information about them
Config	3.0	configure the system
AboutBuild	3.0	get information about the product
Certificate	3.0	handle the certificate management
System	3.0	general control over the system state
ClientControl	3.0	handles the non mode-specific control of the Client including current mode output
ClientRecording	3.2	start/stop recording and visual recording, manage recordings
ClientMap	3.3	start and visualize offline mapping, manage maps

Name of API Module	Version	Description
ClientLocalization	4.0	start/stop and visualize localization
ClientManualAlign	4.1	manually align a constructed map
ClientGlobalAlign	4.0	enables global map consistency using landmarks
ClientLaserMask	3.0	mask out static objects in the field of view of the laser
ClientSensor	3.0	handles the input of sensor data provided by the customer
ClientUser	4.0	manage users on a Localization Client
ServerMap	3.0	manage the central maps and get information about them
ServerUser	4.0	manage users on a Map Server
ServerInternal	1.0	storage of internal network configurations for fleet management(internal use)
SupportReport	3.1	provides system facilities used for customer support
SupportRecovery	3.0	manage system recovery, backup and update

3 Interfaces

The Laser Localization Software provides two main interface types:

1) **RPC Interfaces** based on JSON-RPC communication protocol version 2.0, where queries are sent to and answered by a HTTP (HTTPS) server. Through RPC Interfaces of either Localization Client or Map Server it is possible to control the modes, handle maps, set or retrieve parameters etc.

2) **Binary Interfaces** offering simple and lean communication for low-latency and high-throughput uses. For example, Binary Interfaces handle inputs from external sensors or outputs of the localization estimate, maps, or visualizations.

All interfaces are available through TCP/IP networking, both with and without TLS security. Confer to the list of ports utilized by the Laser Localization Software for a list of all interfaces provided by the system.

Furthermore, the Laser Localization Software uses SI units at everywhere unless explicitly and clearly stated otherwise. The application data sent or received by the Laser

Localization Software uses little-endian byte order for integer values. Floating point values are encoded in IEEE-754 data types, unless noted otherwise.

3.1 RPC Interface

Currently the only mode to operate the RPC interface is by sending queries (*call* or *request*). Each such query calls a specific method in the RPC interface, which may alter the state of the system or request a response containing specific information. For each query received, the system will reply with a JSON RPC response. To transmit information, these queries and resulting responses also carry a message. The format of this message is predetermined and depends entirely on the API method being called. This document contains a list of all available RPC methods and their associated messages.

An example for such a request is given below:

3.1.1 Call

```
...
...
Content-Type: application/json; charset=utf-8
{
  "id": <ID>,
  "jsonrpc": "2.0",
  "method": "someQueryMethod",
    "params": {
    "query": {
      QueryType: see message types
    }
  }
}
```

Note that the “query” JSON object *must* be the only property in the “params” object.

3.1.2 Response

If the JSON-RPC query is received and processed successfully, the response will look like this:

```
...
...
Content-Type: application/json; charset=utf-8
{
  "id": <ID>,
  "jsonrpc": "2.0",
  "result": {
    "response": {
      ResponseType: see message types
    }
  }
}
```

The JSON object containing the result from the Laser Localization Software is called “response”. The response object always contains a ResponseCode entry. Even if this entry states an error, the whole response message is sent. In this case other entries of the response message contain uninitialized or default values, which must not be used.

3.1.3 Error

If a JSON-RPC error occurs while communicating or parsing a query, the response will look like this:

```
...
...
Content-Type: application/json; charset=utf-8
{
  "id": <ID>,
  "jsonrpc": "2.0",
  "error": {
    "code": -32700
    "message": "error description";
  }
}
```

The error codes correspond to those described in the JSON-RPC 2.0 specification.

As a security feature input validation is activated. The user receives the error code -32602 as INVALID_PARAMS for failed input validation. The content of the message is

not conform to the message specification. If the Laser Localization Software sends messages violating the input validation the error code is -32603 for INTERNAL_ERROR.

3.1.4 Message Naming Conventions

The JSON “query” and “response” properties referenced above each contain a Message JSON object. These objects follow certain naming conventions:

1. Any JSON object transmitted directly within the “query” and “response” fields carries a *Message* suffix. Within this API, such JSON objects are called *Message Objects* or just *Messages* for short.
2. JSON objects without the *Message* suffix are never transmitted directly, but only as properties within a Message Object.
3. Messages sent as part of a JSON-RPC “response” always carry the suffix ResponseMessage. They always contain a ResponseCode.

3.1.5 Session IDs

In general, RPC interfaces are session-based and require the user to log in before accessing them. Sessions are identified through unique Session IDs. When calling most RPC methods, such an ID must be sent as part of the query Message. Note that some exceptions to this exist, the most notable being the sessionLogin call.

To keep old sessions from accumulating, every session known to the Localization Client, Map Server, or Support Components has a limited lifespan called the timeout. A given session will expire if it remains unused for longer than its timeout. However, this timer can be reset through an API call. It will also automatically refresh every time the session ID is used within a valid JSON-RPC request. Consequently, a session can be kept alive indefinitely while it remains in use.

A maximum of 500 Session IDs is allowed before getting the response code `SESSION_MAXIMUM_NUMBER_REACHED`, as defined in Session Specific ResponseCodes

3.1.6 Remarks on the HTTP Server

Currently, the HTTP server does not support the `Expect: 100-continue` behavior. Please ensure that the `Expect:` HTTP request header is empty, i.e., use `Expect:`.

3.2 Binary Interface Concept

A Binary Interface is used to communicate in a lean and easy way. It supports no bidirectional communications. Connections to the Binary Interfaces are not session-bound and can be used independently from the RPC Interface. However, the RPC Interface may be used to configure some of the Binary Interfaces. It is *not* possible to submit any kind of configuration or control calls through these interfaces.

In practice, the Binary Interfaces use a simple unidirectional push protocol:

1. The Binary Interfaces communicate through TCP/IP sockets, with or without TLS security.
2. Each Binary Interface listens on two TCP ports: one of them is used exclusively for TLS over TCP, while the other provides unauthorized, unauthenticated and unencrypted communication.
3. Communication is done through packed (unaligned and unpadded) structures called *Datagrams*. Datagrams may be regarded as the equivalent of Messages in the RPC Interface.
4. Each interface is associated with exactly one such Datagram, and will only send or accept that specific structure.
5. The sockets used by the interfaces are configured to minimize delay. Please note that Nagle's algorithm is not active on these sockets due to the TCP_NODELAY option and overall performance of your interface could suffer.
6. Push providers send data from the system by acting as a server; after connecting to such a socket, the user will immediately start to receive Datagrams.
7. Push consumers receive data for the system by connecting to an external provider operated by the user; after such a socket is opened, the user may immediately start to send Datagrams.
8. If input validation fails the binary interface connection will be closed.

3.2.1 Push Provider

A Push Provider, i.e., the TCP server, listens to two ports on which a new TCP connection can be established. The server enforces the use of TLS over TCP on one of these ports. The second port provides unauthorized, unauthenticated and unencrypted communication.

3.2.2 Push Consumer

Analogously to push providers, a push consumer acts as a TCP client. A push consumer will connect to a given TCP server and then receive data provided by the user. Such a consumer thus must know which TCP server to connect to, and whether or not to use TLS security. This information can be specified using the RPC Interface. Note that none of the push consumers are necessary for the initial system setup or usage. Instead, push consumers are used only to receive external sensor data, such as odometry information.

3.2.3 Datagram Naming Conventions

Data transmitted via Binary Interfaces is conveyed as datagrams, where each Datagram defines exactly one entity of coherent data. These datagrams follow certain naming conventions:

1. Any object transmitted carries the suffix *Datagram*.
2. Binary objects without the *Datagram* suffix are never transmitted directly, but only as properties within a Datagram.

4 Communication Security

4.1 Concept

Communication security is offered via the use of TLS over TCP as the underlying communication layer. In the case of the Binary Interface, this means that the TCP connections involved require a TLS handshake before providing or accepting any data. If this handshake is not performed, the connection will be closed. In the case of the RPC Interface, communication security is implemented by using HTTPS instead of HTTP.

The authentication of communication participants is done using a non-configurable common name (CN field). Consequently, the software can be deployed with encrypted communications even before completing the certificate management process. This also leads to the following requirements on the certificates:

4.2 Requirements

1. All Localization Clients *must* and initially *will* provide a certificate with `CN=BoschRexrothLaserLocalizationClient`.
2. The Map Server *must* and initially *will* provide a certificate with `CN=BoschRexrothLaserLocalizationServer`.
3. The graphical user interface and each customer and sensor provider *must* provide a certificate with `CN=BoschRexrothLaserLocalizationUser`.
4. The certificates used in both Map Server and Localization Client *must* provide the field `X509v3 Subject Alternative Name: DNS:localhost, IP:127.0.0.1`. This is due to internal requirements of the product.
5. Localization Client, Map Server and user (including the provided graphical user interface) must be provided with a complete list of necessary root certificate authorities.

4.3 Certificate Management

All requirements will be initially met when the product is delivered (including a certificate-key-pair for the user and a certificate file containing the certificate authority used at production).

To roll out new certificates for both Localization Client and Map Server, use the functionality provided by the Certificates module The user must provide four files in base64 encoding: 1. A root certificate file, e.g. called ca.crt, that contains one or more certificate authority certs. 2. A component certificate file, e.g. called client.crt, that contains the public certificate of the component as defined in the requirements 3. A component key file, e.g. called client.key, that contains the private key paired with the component certificate provided 4. A revocation file, e.g. called revocation.crl, that contains one or more valid certificate revocation lists. For each certificate authority in use, a revocation list MUST be provided in this file.

This certificate rollout process has to be performed for every product element INDIVIDUALLY using the Certificates module. The certificates will be used after the next restart of the system. Note that secure communications can only resume after compatible certificates are rolled out across all product elements.

5 Client Control Mode

The Localization Client offers numerous functionalities, such as recording laser scan, building a map, or localization. To implement these diverse functionalities within a single product element, the Localization Client has **6** operation modes:

- **VISUALRECORDING**: This mode is in *RUN*-state while a map is calculated live during the recording.
- **MAP**: While a map is generated from a recording the *ClientState* of the Map mode is *RUN*.
- **LOC**: This mode is in *RUN*-state while the system localizes itself to a map received from the Map Server.
- **REC**: This mode is in *RUN*-state while a recording is made.
- **ALIGN**: This mode is *RUN*-state while the user works on aligning the map.
- **MASK**: This mode is *RUN*-state while the user is creating a mask for the laser scans.

Each operation mode can have the following states:

Client Control State	Description
INIT	The function is booting up. This state should be transient and it should transit to the next state after a few seconds.
READY	The function is ready but in standby mode. This mode can transit to the state RUN if requested.
RUN	The function is running. This mode is executing the job. When the job is done, the mode transits to the state READY.
NOT_AVAILABLE	The function is not available or out of order.

These client control modes can mostly run in parallel. The only exceptions are MAP, REC, and VISUALRECORDING which are mutually exclusive. While starting up, the client control modes are all in INIT-state. After the startup phase, all modes should be either in READY or RUN. Users may connect to a ClientControlMode Binary Interface to receive information about the current mode.

To request a mode change, the user should call the appropriate methods from the Localization Client's RPC interface. These methods are documented in the chapter Localization Client – RPC Methods. A positive response of these mode-changing methods means only that the request has been received. It does not guarantee that the mode change is successful. Before using any mode-dependent functionality, the user

should thus check the ClientControlMode interface. This way, the user can ensure that the client is actually in the correct mode.

6 Response Codes

6.1 Preface

The RPC interface will respond to queries using a ResponseMessage, which carries a response code. This response code informs the user of the outcome of the query. Response codes are encoded as NonnegativeInteger64. Like the other aspects of the API, these codes are modularized. The most significant 16 bits of each code serve as a module identifier. This identifier indicates the API module to which the given response code belongs. A module identifier of 0 indicates a common response code that is consistent across all modules. The least significant 48 bits indicate the specific response that occurred. Consequently, each numeric response code is uniquely defined across the entire API and all of its modules. Any response code indicating an error also generates a diagnostic entry containing additional error information. The user can retrieve these diagnostic entries through the Diagnostic module

Recall that all methods of the RPC Interface belong to exactly one module. Each method will therefore respond with either a common response code or a code from the module to which the method belongs. This limits the number of response codes which a user has to handle when processing the response of a query.

6.2 Module identifiers

The 16 bit module identifiers are matched to the API modules as follows:

Identifier (16 bit)	Module
0x0000	None (Common)
0x0001	AboutModules
0x0002	Session
0x0003	LicensingFeature
0x0004	Config
0x0005	AboutBuild
0x0006	Certificates
0x0007	System

Identifier (16 bit)	Module
0x0008	Diagnostic
0x0100	ClientRecording
0x0101	ClientMap
0x0102	ClientLocalization
0x0103	ClientManualAlign
0x0104	ClientGlobalAlign
0x0105	ClientLaserMask
0x0106	ClientUser
0x0200	ServerMap
0x0201	ServerUser
0x0300	SupportReport
0x0301	SupportRecovery

6.3 Common Response Codes

These response codes are shared between all modules and thus have a module identifier of 0x0000.

Response code (64 bit, hexadecimal) `RESPONSE`	Description Example
0x0000 0000 0000 0000 `OK`	The operation completed successfully.
0x0000 0000 0000 0001 `WARNING`	The operation completed successfully, but encountered a significant non-fatal issue Information about this issue was written to the diagnostic system.
0x0000 0000 0000 0002 `INTERNAL_ERROR`	An internal system error occurred, which cannot be rectified by the user.
0x0000 0000 0000 0003 `UNKNOWN_ERROR`	A non-specified or ill-formed error occurred.
0x0000 0000 0000 0004 `SESSION_INVALID`	The Session ID given in the request does not match any known session

Response code (64 bit, hexadecimal) `RESPONSE`	Description Example
	A user tries to operate the system without logging in first.
0x0000 0000 0000 0005 `SESSION_EXPIRED`	The session corresponding to the given Session ID has expired.
	A user tries to operate the system after their session has expired due to inactivity.
0x0000 0000 0000 0006 `NOT_AUTHORIZED`	The session corresponding to the given Session ID does not have the necessary permission to perform the operation.
	A user with basic privileges tries to change the password of another user.
0x0000 0000 0000 0007 `NOT_IN_REQUIRED_STATE`	The current system state does not allow the operation to complete.
	A user tried to enable localization while the Localization Client is already busy recording sensor data.
0x0000 0000 0000 0008 `FEATURE_NOT_LICENSED`	The system is not licensed to perform the desired operation.
	A user tries to build a map on a system not licensed for mapping.
0x0000 0000 0000 0009 `INVALID_MESSAGE_CONTENT`	The content of the request message did not conform to the API specifications.
	A parameter sent as part of the message held an invalid value.
0x0000 0000 0000 000a `ENTITY_ALREADY_EXISTS`	Tried to create an entity that already exists.
	A user tried to create a map with a name that is already in use.
0x0000 0000 0000 000b `ENTITY_NOT_FOUND`	Tried to reference an entity does not exist.
	A user tried to delete a map that does not exist.
0x0000 0000 0000 000c `FILE_ACCESS_FAILED`	An error occurred while attempting access a file system.

Response code (64 bit, hexadecimal) `RESPONSE`	Description Example
	The system tried to write a file to a device that has run out of space.
0x0000 0000 0000 000d `SENSOR_NOT_AVAILABLE`	A required sensor was not available. A user tried to record laser scan data while no laser scanner is connected.
0x0000 0000 0000 000e `ENTITY_IN_USE`	Tried to modify an entity that is currently in use by the system. A user tried to delete a recording from which the system is currently building a map.

6.4 *Session*-Specific Response Codes

Response code (64 bit, hexadecimal) `RESPONSE`	Description Example
0x0002 0000 0000 0001 `SESSION_INVALID_CREDENTIALS`	The given user credentials were not valid. A user tried to log in with an incorrect password or username.
0x0002 0000 0000 0003 `SESSION_MAXIMUM_NUMBER_REACHED`	The number of open sessions has reached the predetermined maximum. A user tried to log in while too many existing sessions were already active.
0x0002 0000 0000 0004 `SESSION_TOO_MANY_FAILED_ATTEMPTS`	Login functionality disabled temporarily due to too many failed attempts. A user made too many login attempts with invalid credentials within a short timespan.

6.5 *LicensingFeature*-Specific Response Codes

Response code (64 bit, hexadecimal) `RESPONSE`	Description Example
0x0003 0000 0000 0001 `LICENSING_NO_VALID_LICENSE`	The system has not received any valid license files. A user attempts to retrieve license information without first sending a license.
0x0003 0000 0000 0002 `LICENSING_LICENSE_EXPIRED`	The license has expired. The date of expiration has passed while using a non-permanent license.
0x0003 0000 0000 0003 `LICENSING_METHOD_NOT_AVAILABLE`	The method is not available for the currently selected licensing method. A user calls a licensing API method that is not applicable with the selected licensing method.
0x0003 0000 0001 0000 `LICENSING_DONGLE_ERROR`	The dongle reported an error. A user called a method that failed because of an error within the hardware dongle.
0x0003 0000 0001 0001 `LICENSING_DONGLE_NOT_FOUND`	No licensing dongle was not found while using dongle-based licensing. The hardware dongle is not connected to the host machine.
0x0003 0000 0001 0002 `LICENSING_PREFERRED_DONGLE_NOT_FOUND`	The perferred licensing dongle was not found. A user has specified a preferred dongle that is not connected to the host machine.
0x0003 0000 0001 0003 `LICENSING_MULTIPLE_DONGLES_FOUND`	Multiple licensing dongles were found while using dongle-based licensing. A user connected two or more dongles to the host machine, but didn't specify a preferred dongle.
0x0003 0000 0001 0004 `LICENSING_DONGLE_SERVER_ERROR`	The communication with the dongle server on the host machine has failed. The dongle server software is not running on the host machine.
0x0003 0000 0002 0000 `LICENSING_TPM_ERROR`	The TPM reported an error while using TPM-based licensing.

Response code (64 bit, hexadecimal) `RESPONSE`	Description Example
	A user called a method that failed because of an error in the trusted platform module.
0x0003 0000 0002 0001 `LICENSING_TPM_NOT_FOUND`	No TPM was found while using TPM-based licensing.
	The host system is not equipped with a Trusted Platform Module.
0x0003 0000 0002 0002 `LICENSING_TPM_NOT_SUPPORTED`	An unsupported TPM was found while using TPM-based licensing.
	The TPM installed in the system does not offer the functionality required for licensing.
0x0003 0000 0002 0003 `LICENSING_TPM_CERTIFICATES_NOT_FOUND`	The certificates needed to verify the TPM are missing.
	A user did not specify a TPM verification file when setting up the component.
0x0003 0000 0002 0004 `LICENSING_TPM_VERIFY_FAILED`	The TPM could not be verified using the available certificates.
	During initial setup, a user specified a TPM verification file that does not match the TPM installed in the host.

6.6 *Config*-Specific Response Codes

Response code (64 bit, hexadecimal) `RESPONSE`	Description Example
0x0004 0000 0000 0001 `CONFIG_KEY_INVALID`	A given configuration key was not valid.
	A user attempted to write a configuration entry with a key that does not exist.
0x0004 0000 0000 0002 `CONFIG_VALUE_TYPE_INVALID`	The type of the given value does not match the expected type for this key.
	A user tried to set a config entry with a numeric value type to a character string value.

6.7 *Certificate*-Specific Response Codes

Response code (64 bit, hexadecimal) `RESPONSE`	Description Example
0x0006 0000 0000 0001 `CERTIFICATE_CERT_FILETYPE_INVALID`	Certificate file could not be parsed. The file was not in PEM format or did not contain a valid X.509 certificate.
0x0006 0000 0000 0002 `CERTIFICATE_KEY_FILETYPE_INVALID`	Private key file could not be parsed. The file was not in PEM format or did not contain a valid private key.
0x0006 0000 0000 0003 `CERTIFICATE_AUTH_FILETYPE_INVALID`	Certificate authority file could not be parsed. The file was not in PEM format or did not contain any valid X.509 certificates.
0x0006 0000 0000 0004 `CERTIFICATE_REVLIST_FILETYPE_INVALID`	Certificate revocation list file could not be parsed. The file was not in PEM format or did not contain any valid X.509 client revocation lists.
0x0006 0000 0000 0005 `CERTIFICATE_KEY_INVALID`	Private key could not be used. The private key did not match the provided certificate.
0x0006 0000 0000 0006 `CERTIFICATE_CHAIN_INVALID`	Certificate chain invalid. The certificate authority file does not contain the signing authority of the certificate.
0x0006 0000 0000 0007 `CERTIFICATE_CERT_COMMONNAME_INVALID`	Certificate common name does not match specifications.
0x0006 0000 0000 0008 `CERTIFICATE_CERT_VERIFY_FAILED`	Certificate provided could not be verified. The signing CA could not verify certificate due to a missing CRL entry.

6.8 *ClientMap*-Specific Response Codes

Response code (64 bit, hexadecimal) RESPONSE	Description Example
0x0101 0000 0000 0001 CLIENTMAP_SERVER_UNREACHABLE	The Localization Client could not reach the Map Server while trying to send a map.
0x0101 0000 0000 0002 CLIENTMAP_SERVER_COMMUNICATION_- FAILED	The Localization Client succesfully connected to the Map Server, but there was an error when sending or receiving map data.

6.9 *ClientGlobalAlign*-Specific Response Codes

Response code (64 bit, hexadecimal) RESPONSE	Description Example
0x0104 0000 0000 0001 CLIENTGLOBALALIGN_OBSERVATION_- NOT_FOUND	An observation with the specified Id could not be found. A user tried to delete an observation that does not exist

6.10 *SupportRecovery*-Specific Response Codes

Response code (64 bit, hexadecimal) RESPONSE	Description Example
0x0301 0000 0000 0001 SUPPORTRECOVERY_VERSION_INCOMPAT- IBLE	The recovery point has an incompatible version while trying to restore from this recovery point.
0x0301 0000 0000 0002 SUPPORTRECOVERY_READ_ARCHIVE_- FAILED	The system failed to extract or decrypt information from the recovery point.

Response code (64 bit, hexadecimal) `RESPONSE`	Description Example
0x0301 0000 0000 0003 `SUPPORTRECOVERY_RECOVERY_FAILED`	The system failed to recover the data from the recovery point.

7 Common JSON Objects and Types

The following common JSON objects and types are used across multiple messages used by the RPC interfaces. Within the context of this section, the types “integer”, “number”, “boolean”, “string”, and “array” refer to the basic JSON data types defined in RFC 8259.

7.1 Types

7.1.1 AsciiCharacterString (string)

A UTF-8 encoded character string, which contains only characters that can be encoded by a single byte which is within the range of [0x20, 0x7e]. This corresponds to those characters within the ASCII standard that can be printed directly, such as a-z, A-Z or 0-9. An AsciiCharacterString must not include any other control characters, such as the *newline* (UTF-8: 0x0a) or *carriage return* (UTF-8: 0x0d) character.

7.1.2 EmptyString (string)

A string with a fixed length of 0.

7.1.3 Integer64 (integer)

A two-complement signed integer of 64 bits, in decimal notation, with a possible value within the interval $[-2^{63}, 2^{63})$ or [-9223372036854775808, 9223372036854775807].

7.1.4 NonnegativeInteger64 (integer)

An Integer64 that excludes negative values. Thus, the allowed interval is [0, 2^{63}) or [0, 9223372036854775807].

7.1.5 IEEE754Double (number)

A rational number in decimal notation, with a range and precision that is, at most, as good as that provided by the IEEE754 double-precision format. Note that due to the textual representation used by JSON, some loss of precision must be expected. See RFC 8259 for details. Values must lie within the interval [-1.797693134862 * 10308, 1.797693134862 * 10308]. This data type does not accept special values allowed under IEEE754, such as NaN or Infinity.

7.1.6 ResponseCode (NonnegativeInteger64)

Response codes are given as non-negative integers. Refer to the relevant section for details on their interpretation.

7.1.7 SessionId (string)

Session IDs are RFC4122-compliant Universally Unique IDentifiers (UUIDs). They are encoded as a UTF-8 encoded string that contains the UUID in hexadecimal notation. This string has the form *XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX*, where *X* is a digit from 0 to 9 or a letter from a to f. Note that UUIDs are case-insensitive, and that letters contained within them can thus be given in lower or upper case.

7.1.8 Size (NonnegativeInteger64)

Discrete sizes and counts are encoded as non-negative integers.

7.1.9 RecordingName (AsciiCharacterString, length > 0)

Recordings are referenced by non-empty strings of up to 240 characters in length. These names may not contain forward slashes (‘/’). Additionally, the strings ”.” and ”..” are not valid recording names.

7.1.10 ClientMapName (AsciiCharacterString, length > 0)

Client-side maps are referenced by non-empty strings of up to 240 characters in length. These names may not contain forward slashes (‘/’). Additionally, the strings ”.” and ”..” are not valid map names.

7.1.11 ServerMapName (AsciiCharacterString, length > 0)

Server-side maps are referenced by non-empty strings of up to 240 characters in length. These names may not contain forward slashes (‘/’). Additionally, the strings ”.” and ”..” are not valid map names.

7.1.12 NetworkAddress(AsciiCharacterString, length > 0)

A non-empty string representing a valid network address (IP address or hostname) referring to a target host. May include a colon-separated destination port.

7.2 Objects

7.2.1 Timestamp

- “valid”: boolean
- “time”: Integer64
- “resolution”: number, must be one of 1, 100, 1000, 1000000, or 1000000000

Represents a point in time at which a given event occurred. “time” gives the number of complete ticks that have passed since 00:00:00 UTC on Thursday, 1 January 1970. “resolution” specifies the positive number of ticks that make up one SI second (e.g., 1000000 for ticks equivalent to microseconds). If “valid” is set to false, the timestamp has no meaning and “value” and “resolution” are undefined.

7.2.2 TimeInterval

- “valid” : boolean
- “time” : Integer64
- “resolution” : number, must be one of 1, 100, 1000, 1000000, or 1000000000

Represents the interval of time that elapsed between two events. “time” gives the number of complete ticks that have passed between the two events. “resolution” specifies the positive number of ticks that make up one SI second (e.g., 1000000 for ticks equivalent to microseconds). If “valid” is set to false, the time interval has no meaning and “value” and “resolution” are undefined.

7.2.3 Pose3D

- “x”: IEEE754Double
- “y”: IEEE754Double
- “z”: IEEE754Double
- “qw”: IEEE754Double
- “qx”: IEEE754Double
- “qy”: IEEE754Double
- “qz”: IEEE754Double

Describes the pose of an object within a 3D coordinate system, given by the object position “x”, “y”, “z” and the orientation quaternion “qw”, “qx”, “qy”, and “qz”.

7.2.4 Transform3D

- “x”: IEEE754Double
- “y”: IEEE754Double
- “z”: IEEE754Double
- “qw”: IEEE754Double
- “qx”: IEEE754Double
- “qy”: IEEE754Double
- “qz”: IEEE754Double

Describes a relative transformation between two 3D coordinate systems, given by the translation “x”, “y”, “z” and the rotation quaternion “qw”, “qx”, “qy”, and “qz”.

7.2.5 Pose2D

- “x”: IEEE754Double
- “y”: IEEE754Double
- “a”: IEEE754Double

Describes the pose of an object within a 2D coordinate system, given by the object position “x”, “y” and the orientation angle “a”.

7.2.6 Transform2D

- “x”: IEEE754Double
- “y”: IEEE754Double
- “a”: IEEE754Double

Describes a relative transformation between two 2D coordinate systems, given by the translation “x”, “y” and the rotation angle “a”.

7.2.7 Container

- “contentEncoding”: “base64” (string)
- “contentMediaType”: AsciiCharacterString (length > 0)
- “content”: AsciiCharacterString containing only those base64 encoding characters specified in RFC 4648

The RPC interfaces use containers to transmit binary data via JSON. The data is encoded in RFC4648-compliant base64 and stored within the “content” string (see FAQ for example). “contentMediaType” is a non-empty string that describes the type of media which the binary data encodes, e.g., “image/png” for a PNG image.

Media Type “application/PointCloud2D” A container of this type holds a 2D point cloud as a base64-encoded (see FAQ for example) byte-array of length `2 * 4 * pointCloudSize`. To construct the point cloud, reinterpret this array as `2 * pointCloudSize` 32-bit IEEE-754 single-precision floating-point values. Each point is then specified by two consecutive such values, which give the point’s Cartesian coordinates (x,y).

Media Type “image/png” A container of this media type holds a base64-encoded PNG image (see FAQ for example).

Media Type “application/LicenseFile” A container with this media type holds the content of a binary license key file in base64 encoding.

Media Type "application/CertificateFile" A container with this media type holds the content of a binary certificate file in base64 encoding.

8 Shared (Localization Client / Map Server / Support) – RPC Methods

The RPC methods listed in this section currently exist for both Localization Client and Map Server and are therefore only stated once in this document. Note that in future versions, the methods may differ for the Localization Client and Map Server. In this case, the methods will be split into client and server variants and the documentation will be updated accordingly.

8.1 API Module Information (Module: *AboutModules*)

8.1.1 aboutModulesList

This call is used to query a list of the used API modules and their versions.

- QueryType: AboutModulesQueryMessage
- ResponseType: AboutModulesListResponseMessage

Query Value None (empty JSON object)

Response Value

- response code as defined in the relevant section
- array of AboutModulesModule, containing the available modules and their versions

8.1.2 Messages

AboutModulesQueryMessage This message has no properties, and consists of an empty JSON object.

AboutModulesListResponseMessage

- "responseCode": ResponseCode
- "modules": array of AboutModulesModule

8.1.3 Objects

AboutModulesModule

- "name": name of the module, as specified in the API module overview.
- "majorVersion": the module's major version number.
- "minorVersion": the module's minor version number.

8.2 Session Control (Module: *Session*)

8.2.1 sessionLogin

Login to a product element

- QueryType: SessionLoginMessage
- ResponseType: SessionLoginResponseMessage

Query Value

- username and password of the user requesting to be logged in
- timeinterval: This is the duration after which an inactive session will expire if it has not been refreshed. A session with a timeout of 0 or an invalid timeout will never expire.

Response Value

- response code as defined in the relevant section
- sessionId for the newly established session, or an empty string in case of a sessionLogin failure

8.2.2 sessionLogout

Logout from a product element

- QueryType: SessionQueryMessage
- ResponseType: SessionResponseMessage

Query Value

- sessionId: the Session ID to log out

Response Value

- response code as defined in the relevant section

8.2.3 sessionRefresh

Refreshes the given session, resetting its expiration timer. This works as a heartbeat mechanism, allowing a session to be kept alive even if it is not being used

- QueryType: SessionQueryMessage
- ResponseType: SessionResponseMessage

Query Value

- sessionId: the Session ID to refresh

Response Value

- response code as defined in the relevant section

8.2.4 sessionGroupInfo

Requests the user groups to which the session's user belongs.

- QueryType: SessionQueryMessage
- ResponseType: SessionGroupsResponseMessage

Query Value

- sessionId: the Session ID to refresh

Response Value

- response code as defined in the relevant section
- array of user groups of the current session (array should have exactly the size 1, since the user can only be member in one group)

8.2.5 Messages

SessionLoginMessage

- “timeout”: TimeInterval
- “userName”: AsciiCharacterString
- “password”: AsciiCharacterString

SessionLoginResponseMessage

- “responseCode”: ResponseCode
- “sessionId”: SessionId or EmptyString

SessionQueryMessage

- “sessionId”: SessionId

SessionResponseMessage

- “responseCode”: ResponseCode

SessionGroupsResponseMessage

- “responseCode”: ResponseCode
- “userGroups”: array of AsciiCharacterString

8.3 Diagnostic Information (Module: *Diagnostic*)

8.3.1 diagnosticList

Retrieves all available diagnostic entries from the buffer

- QueryType: DiagnosticQueryMessage
- ResponseType: DiagnosticArrayResponseMessage

Query Value

- sessionId: the session requesting diagnostic information

Response Value

- “responseCode”: ResponseCode
- array of all diagnostic messages available as per the permission of the given session

8.3.2 diagnosticClear

Clears the diagnostic buffer, removing all stored diagnostic entries

- QueryType: DiagnosticQueryMessage
- ResponseType: DiagnosticResponseMessage

Query Value

- sessionId: the session requesting to clear diagnostic information

Response Value

- “responseCode”: ResponseCode

8.3.3 Messages

DiagnosticQueryMessage

- “sessionId”: SessionId

DiagnosticResponseMessage

- “responseCode”: ResponseCode

DiagnosticArrayResponseMessage

- “responseCode”: ResponseCode
- “diagnosticEntries”: array of DiagnosticEntry

8.3.4 Objects

DiagnosticEntry

- “id”: Integer64
- “componentName”: AsciiCharacterString
- “diagnosticCode”: Integer64
- “diagnosticName”: AsciiCharacterString
- “diagnosticText”: AsciiCharacterString
- “timestamp”: Timestamp
- “additionalInfo”: array of AsciiCharacterString

A complete list of error diagnostic names and English description strings will be included in one of the next versions of the documentation.

Some diagnostic messages contain online generated information, for these messages the diagnosticText contains placeholders, e.g. {sessionId}. In parallel the additionalInfo, an array with even number of elements contains pairwise the placeholder and the actual value.

diagnosticText: ’New session {sessionId} for a member of the group {group}’ additionalInfo: [’group’,’admin’,’sessionId’,’4f518’]

The assembled message is ’New session 4f518 for a member of the group admin’.

The diagnosticCode is a 64-bit bitfield which specifies the type of the given DiagnosticEntry. Only the three most significant bits (bits 63-61) are currently used by the API, while the remaining bits (bits 60-0) are reserved for internal use. These three most significant bits specify the type of entry according to the following table:

Entry type	Value (3 bit unsigned integer)
INFO	6
WARNING	4

Entry type	Value (3 bit unsigned integer)
ERR	3
CRIT	2

Refer to the Laser Localization Software manual for the meaning of these entry types.

8.4 Software Licensing (Module: *LicensingFeature*)

A software license is needed to unlock each product element's full functionality. Licenses are provided as digital software license files, which are bound to specific host machines. Host machines are identified by their host ID, as provided by the licensingFeatureGetHostId method. Section 14.1 lists the available methods for host machine identification. After aquiring a license for a given host the associated license file must be sent to the product element using the licensingFeatureSet method.

Note: This section merely describes the API functionality used to license the Bosch Rexroth Laser Localization Software. Refer to the Licensing section for information on what licenses are required to use the other functionality described in this document.

8.4.1 licensingFeatureGetTrustedPlatformModuleInformation

Retrieves the Trusted Platform Module Information of the host.

- QueryType : LicensingQueryMessage
- ResponseType : LicensingHostTpmResponseMessage

Query Value

- sessionId: the session requesting the TPM information

Response Value

- response code as defined in the relevant section
- Trusted Platform Module information

8.4.2 licensingFeatureGetHostId

Retrieves the hardware-specific host ID.

- QueryType : LicensingQueryMessage
- ResponseType : LicensingHostIdResponseMessage

Query Value

- sessionId: the session requesting the host ID

Response Value

- response code as defined in the relevant section
- host ID

8.4.3 licensingFeatureSet

Sends a software license to the product element.

- QueryType : LicensingKeyMessage
- ResponseType : LicensingResponseMessage

Query Value

- sessionId: the session uploading the license key
- licenseKey: a base64 encoded container carrying the software license file to be sent

Response Value

- response code as defined in the relevant section

8.4.4 licensingFeatureList

Retrieves information about the current license key.

- QueryType : LicensingQueryMessage
- ResponseType : LicensingInfoResponseMessage

Query Value

- sessionId: the session requesting the license info

Response Value

- response code as defined in the relevant section
- issue date of the license
- expiry date of the license
- name of the product vendor
- host id, the license is issued to this host id
- license version
- license status
- license type
- list of features available in the license

8.4.5 Messages

LicensingQueryMessage

- “sessionId”: SessionId

LicensingHostIdResponseMessage

- “responseCode”: ResponseCode
- “hostId”: AsciiCharacterString

LicensingHostTpmResponseMessage

- “responseCode”: ResponseCode
- “hardwareInformation”: AsciiCharacterString or EmptyString
- “certificateIssuer”: AsciiCharacterString or EmptyString
- “certificateSubject”: AsciiCharacterString or EmptyString
- “certificateAuthorityKeyIdentifier”: AsciiCharacterString or EmptyString
- “certificateSubjectAlternativeNames”: array of AsciiCharacterString or array of EmptyString
- “certificateAuthorityInformationAccess”: array of AsciiCharacterString or array of EmptyString

LicensingKeyMessage

- “sessionId”: SessionId
- “licenseKey”: Container of media type application/LicenseFile.

LicensingResponseMessage

- “responseCode”: ResponseCode

8.4.6 Objects

LicensingFeature

- “featureName”: AsciiCharacterString
- “featureVersion” : AsciiCharacterString
- “featureCapability” : one of the possible license feature capabilities defined in the enumeration LicensingFeatureCapability

LicensingInfoResponseMessage

- “responseCode”: ResponseCode
- “issueDate”: timestamp
- “expiryDate” : timestamp
- “vendorName” : AsciiCharacterString
- “hostId” : AsciiCharacterString
- “licenseVersion” : AsciiCharacterString
- “licenseStatus” : one of the possible license statuses defined in the enumeration LicensingStatus
- “licenseType” : one of the possible license types defined in the enumeration LicensingType
- “featureList” : array of LicensingFeature

LicensingStatus

- “Valid”: 1
- “Invalid” : 2
- “Not available” : 4
- “Unknown” : 8

LicensingType

- “Perpetual”: 1
- “Feature based” : 2
- “Time based” : 4
- “Trial” : 8
- “Volume license key” : 16
- “Pay per use, post paid” : 32
- “Pay per use, pre paid” : 64
- “Unknown” : 8

LicensingFeatureCapability

- “Feature localization”: 1
- “Feature mapping” : 2
- “Feature live SLAM” : 3
- “Feature map server” : 4
- “Feature global alignment” : 5

8.5 System Configuration (Module: *Config*)

8.5.1 configList

Returns a list of all configurable parameters, as well as their current values.

- QueryType: ConfigQueryMessage
- ResponseType: ConfigEntryArrayResponseMessage

Query Value

- sessionId: the session requesting the config

Response Value

- response code as defined in the relevant section
- all config entries that are visible for the current user with his sessionId

8.5.2 configSet

Sets all configurable parameters to the given values.

While updating the parameters, the component may reject additional API method calls with a `NOT_IN_REQUIRED_STATE` response. The component will be ready for further method calls once configSet has returned. Note that parameter changes may not take effect if they concern a Client Control Mode that is currently in the RUN state, e.g. while localization is active. This API method should therefore only be used when the Client Control Modes are in the READY state.

- QueryType: ConfigEntryArrayQueryMessage
- ResponseType: ConfigResponseMessage

Query Value

- sessionId: the session setting the config
- all config entries that should be rewritten to the given values

Response Value

- response code as defined in the relevant section

8.5.3 Messages

ConfigEntryArrayMessage

- “sessionId”: SessionId
- “configEntries”: array of ConfigEntry

ConfigResponseMessage

- “responseCode”: ResponseCode

ConfigQueryMessage

- “sessionId”: SessionId

ConfigEntryArrayResponseMessage

- “responseCode”: ResponseCode
- “configEntries”: array of ConfigEntry

8.5.4 Objects

ConfigEntry

- “key”: AsciiCharacterString
- “value”: any of ConfigEntries

ConfigEntries The following tables show the key-value pairs as ConfigEntry for the Laser Localization Software

Map Server Configuration Paramters	Type	Description	Permission
mu_server. changesUntilUpdate	NonnegativeInteger64	number of changed segments received from the Localization Client(s) until a new updated map is generated and published	user, admin
licensing.method	AsciiCharacterString	method used to determine the systems host id (possible values see Section 14.1)	user, admin
licensing.wibudongle. preferredDongleSerial	AsciiCharacterString	use the given dongle serial number for host id, especially if multiple dongles are available	user, admin
map_update.local_ dynamic_map_source. storeMapHistory	boolean	enable saving map history to disk	admin
supportrecovery.autosave. enable	boolean	enable autosave for support report	admin
supportrecovery.autosave. intervalInDays	NonnegativeInteger64	time interval between two autosaves; must be larger than 0	admin

Map Server Configuration Paramters	Type	Description	Permission
supportrecovery.autosave.numberOfPoints	NonnegativeInteger64	number of autosave points to keep; if zero all autosaves are removed	admin

Localization Client Configuration Paramters	Type	Description	Permission
application.vehicleName	AsciiCharacterString	Vehicle name that is used to recognize specific vehicles in the Map Server logs	user, admin
application.mapServerHost	NetworkAddress	Address (client resolvable netname of ip address) of the Map Server, without a port	user, admin
application.localization.autostart	boolean	Determines whether localization begins immediately after starting the Localization Client	user, admin
application.localization.activeMapName	ServerMapName	Name of the map to use on this Localization Client; must match a map name on the Map Server	user, admin
config_laser.shall_mirror_scans	boolean	Flag that indicates if the laser is mounted upside down	user, admin
config_laser.mask.max_range	IEEE754Double	Laser ranges above this value will be ignored	user, admin
config_laser.mask.min_range	IEEE754Double	Laser ranges below this value will be ignored	user, admin

Localization Client Configuration Paramters	Type	Description	Permission
config_laser.mask.min_range_line_data	array of IEEE754Double	(x1, y1, x2, y2, ...) every 4 numbers represent a line; laser beams intersecting these lines will be ignored if the range is *below* the intersection point	user, admin
config_laser.mask.max_range_line_data	array of IEEE754Double	(x1, y1, x2, y2, ...) every 4 numbers represent a line; laser beams intersecting these lines will be ignored if the range is *above* the intersection point	user, admin
LaserComponent.laserType	string	Name of the laser driver to use: omron, sicklms, sickcola2, sickmics3, idec, pfr2000, simple, simpletls. See table below for details.	user, admin
LaserComponent.laserAddress	NetworkAddress	Address of the laser in question. See table below for details.	user, admin
ExternalSensorComponent.Odometry.enabled	boolean	Determines if an external odometry source should be used	user, admin
ExternalSensorComponent.Odometry.address	NetworkAddress	Address `<client resolvable netname of ip address>:<port>` of the external odometry source	user, admin
ExternalSensorComponent.Odometry.tls	boolean	Determines if TLS should be used when connecting to the external odometry source	user, admin

Localization Client Configuration Paramters	Type	Description	Permission
ExternalSensorComponent.InertialMeasurementUnit.enabled	boolean	Determines if an external inertial measurement unit should be used	user, admin
ExternalSensorComponent.InertialMeasurementUnit.address	NetworkAddress	Address `<client resolvable netname of ip address>:<port>` of the external inertial measurement unit	user, admin
ExternalSensorComponent.InertialMeasurementUnit.tls	boolean	Determines if an external inertial mesaurement unit should be used	user, admin
global_strategies.laserscan_frontend_imu_odo_mode	AsciiCharacterString	Set the frontend strategy to incorporate the odometry and IMU data: NONE, ODO_ONLY, IMU_ODO	user, admin
locator_graph.odom_error_model.laser_T_odo.{x,y,z,roll,pitch,yaw}	IEEE754Double	Coordinates of the platform center of rotation with respect to the laser reference frame used in the odometry error model	user, admin
locator_graph.odom_error_model.laser_T_odo.enabled	boolean	enable the usage of these coordinates instead of those in odometry_handler.laser_T_odo if the center of rotation and the odometry reference frame do not match	user, admin
locator_graph.imu_error_model.laser_T_imu.{x,y,z,roll,pitch,yaw}	IEEE754Double	Coordinates of the platform center of rotation with respect to the laser reference frame used in the IMU error model	user, admin

Localization Client Configuration Paramters	Type	Description	Permission
locator_graph.imu_error_model.laser_T_imu.enabled	boolean	enable the usage of these coordinates instead of those in imu_handler.laser_T_-imu if the center of rotation and the IMU reference frame do not match	user, admin
odometry_handler.laser_T_odo.{x,y,z,roll,pitch,yaw}	IEEE754Double	Coordinates of the odometry unit's reference frame with respect to the laser reference frame	user, admin
imu_handler.laser_T_imu.{x,y,z,roll,pitch,yaw}	IEEE754Double	Coordinates of the IMU's reference frame with respect to the laser reference frame	user, admin
licensing.method	AsciiCharacterString	method used to determine the systems host id (possible values see Section 14.1)	user, admin
licensing.wibudongle.preferredDongleSerial	AsciiCharacterString	use the given dongle serial number for host id, especially if multiple dongles are available	user, admin
multi_hypotheses_localization.map_update_info.enable	boolean	enable map update	admin
application.localization.mapHistoryDuration	NonnegativeInteger64	duration of map history in seconds	admin
ltrComponent.enableSupportDataRecorder	boolean	enable support data recording	admin
ltrComponent.supportDataDuration_h	NonnegativeInteger64	duration of recording data for support in hours	admin
supportrecovery.autosave.enable	boolean	enable autosave for support report	admin

Localization Client Configuration Paramters	Type	Description	Permission
supportrecovery.autosave.intervalInDays	NonnegativeInteger64	time interval between two autosaves; must be larger than 0	admin
supportrecovery.autosave.numberOfPoints	NonnegativeInteger64	number of autosave points to keep; if zero all autosaves are removed	admin

The parameter `LaserComponent.laserType` governs the laser driver to be used by the Localization Client. Note that the format of the `LaserComponent.laserAddress` depends on the driver in use. For example, this may be an IP address and port in the format `<client resolvable netname or ip address>:<port>`, such as `192.168.0.10:2112`. Please refer to the Laser Localization Software manual for details on supported laser scanners as well as their associated `LaserComponent.laserType` and `LaserComponent.laserAddress` settings.

8.6 System Information (Module: *AboutBuild*)

8.6.1 aboutBuildList

Returns human-readable version information.

- QueryType: AboutQueryMessage
- ResponseType: AboutResponseMessage

Query Value

- sessionId: the session requesting the information

Response Value

- responseCode: Response code as defined in the relevant section
- aboutString: Human-readable information on versions of Localization Client/Map Server and core components.

8.6.2 Messages

AboutQueryMessage

- “sessionId”: SessionId

AboutResponseMessage

- “responseCode”: ResponseCode
- “aboutString”: string

8.7 Certificate Management (Module: *Certificates*)

8.7.1 certificateSet

Sets the certificates for the communication between the Map Server/Localization Client and the software implemented by the customer. The new certificates are used only after restarting the product element in question.

- QueryType: CertificateQueryMessage
- ResponseType: CertificateResponseMessage

Query Value

- sessionId: the session requesting the information
- root: the container holding the root certificate
- crt: the container holding the certificate
- key: the container holding the key
- rev: the container holding the revocation list

Response Value

- response code as defined in the relevant section

8.7.2 Messages

CertificateQueryMessage

- “sessionId”: SessionId
- “root”: Container of media type application/CertificateFile
- “crt”: Container of media type application/CertificateFile
- “key”: Container of media type application/CertificateFile
- “rev”: Container of media type application/CertificateFile

CertificateResponseMessage

- “responseCode”: ResponseCode

8.8 System Control (Module: *System*)

8.8.1 systemShutdown

The product element receiving this message is requested to shut down. Depending on the system configuration, the element may automatically restart afterwards.

- QueryType: SystemQueryMessage
- ResponseType: SystemResponseMessage

Query Value

- sessionId: the session requesting the shutdown

Response Value

- response code as defined in the relevant section

8.8.2 Messages

SystemQueryMessage

- “sessionId”: SessionId

SystemResponseMessage

- “responseCode”: ResponseCode

9 Localization Client – RPC Methods

9.1 Recording (Module: *ClientRecording*)

9.1.1 clientRecordingStart

Requests to start recording laser scans of the environment, which can later be assembled into a map.

- QueryType: ClientRecordingNameMessage
- ResponseType: ClientRecordingResponseMessage

Query Value

- sessionId: the session requesting the mode change
- name of the new recording (must be unique or an error is thrown)

Response Value

- response code as defined in the relevant section

9.1.2 clientRecordingStop

Requests to stop recording data.

- QueryType: ClientRecordingQueryMessage
- ResponseType: ClientRecordingResponseMessage

Query Value

- sessionId: the session requesting the mode change

Response Value

- response code as defined in the relevant section

9.1.3 clientRecordingStartVisualRecording

Requests to start recording laser scans of the environment, which can later be assembled into a map. The system also constructs a temporary map, which visualizes the areas that have already been recorded.

- QueryType: ClientRecordingNameMessage
- ResponseType: ClientRecordingResponseMessage

Query Value

- sessionId: the session requesting the mode change
- name of the new recording (must be unique or an error is thrown)

Response Value

- response code as defined in the relevant section

9.1.4 clientRecordingStopVisualRecording

Requests to stop recording data and to stop the construction of a temporary map.

- QueryType: ClientRecordingQueryMessage
- ResponseType: ClientRecordingResponseMessage

Query Value

- sessionId: the session requesting the mode change

Response Value

- response code as defined in the relevant section

9.1.5 clientRecordingList

Retrieves a list of all recordings stored by the Localization Client.

- QueryType: ClientRecordingQueryMessage
- ResponseType: ClientRecordingListResponseMessage

Query Value

- sessionId

Response Value

- response code as defined in the relevant section
- new list of managed recordings

9.1.6 clientRecordingRename

Renames a recording stored by the Localization Client.

- QueryType: ClientRecordingRenameMessage
- ResponseType: ClientRecordingResponseMessage

Query Value

- sessionId
- current name of the recording
- desired new name of the recording

Response Value

- response code as defined in the relevant section

9.1.7 clientRecordingDelete

Deletes a recording stored by the Localization Client.

- QueryType: ClientRecordingNameMessage
- ResponseType: ClientRecordingResponseMessage

Query Value

- sessionId
- name of the recording to be deleted

Response Value

- response code as defined in the relevant section

9.1.8 clientRecordingSetCurrentPose

Provides the Localization Client with the current pose of the laser sensor while recording. This pose must be determined through some external means, such as by moving the platform to known reference position.

- QueryType: ClientRecordingPoseMessage
- ResponseType: ClientRecordingResponseMessage

Query Value

- sessionId
- the pose of the laser sensor at the time this method is called

Response Value

- response code as defined in the relevant section

9.1.9 Messages

ClientRecordingQueryMessage

- “sessionId”: SessionId

ClientRecordingNameMessage

- “sessionId”: SessionId
- “recordingName”: RecordingName

ClientRecordingRenameMessage

- “sessionId”: SessionId
- “currentRecordingName”: RecordingName
- “desiredRecordingName”: RecordingName

ClientRecordingPoseMessage

- “sessionId”: SessionId
- “pose”: Pose2D

ClientRecordingListResponseMessage

- “responseCode”: ResponseCode
- “recordingNames”: array of RecordingName

ClientRecordingResponseMessage

- “responseCode”: ResponseCode

9.2 Client-Side Map Management (Module: *ClientMap*)

9.2.1 clientMapStart

Requests to create a map from the data contained within a recording.

- QueryType: ClientMapMappingMessage
- ResponseType: ClientMapResponseMessage

Query Value

- sessionId: the session requesting the mode change
- name of the stored recording used to build the map
- name of the client-side map that should be created (must be unique or an error is thrown)

Response Value

- response code as defined in the relevant section

9.2.2 clientMapStop

Requests to stop map creation.

- QueryType: ClientMapQueryMessage
- ResponseType: ClientMapResponseMessage

Query Value

- sessionId: the session requesting the mode change

Response Value

- response code as defined in the relevant section

9.2.3 clientMapList

Retrieves all completed maps stored by the Localization Client.

- QueryType: ClientMapQueryMessage
- ResponseType: ClientMapListResponseMessage

Query Value

- sessionId: the session requesting the map list

Response Value

- response code as defined in the relevant section
- list of managed client-side maps

9.2.4 clientMapRename

Renames a map stored by the Localization Client.

- QueryType: ClientMapRenameMessage
- ResponseType: ClientMapListResponseMessage

Query Value

- sessionId: the session requesting the map rename
- current name of the client-side map
- desired new name of the client-side map

Response Value

- response code as defined in the relevant section

9.2.5 clientMapDelete

Deletes a map stored by the Localization Client.

- QueryType: ClientMapNameMessage
- ResponseType: ClientMapListResponseMessage

Query Value

- sessionId: the session requesting the map deletion
- name of the client-side map

Response Value

- response code as defined in the relevant section

9.2.6 clientMapSend

Requests the Localization Client to send the map to the Map Server

- QueryType: ClientMapNameMessage
- ResponseType: ClientMapResponseMessage

Query Value

- sessionId: the session requesting the map upload
- name of the client-side map that should be sent to the Map Server

Response Value

- response code as defined in the relevant section

9.2.7 Messages

ClientMapMappingMessage

- “sessionId”: SessionId
- “recordingName”: RecordingName
- “clientMapName”: ClientMapName

ClientMapQueryMessage

- “sessionId”: SessionId

ClientMapNameMessage

- “sessionId”: SessionId
- “clientMapName”: ClientMapName

ClientMapRenameMessage

- “sessionId”: SessionId
- “currentClientMapName”: ClientMapName
- “desiredClientMapName”: ClientMapName

ClientMapResponseMessage

- “responseCode”: ResponseCode

ClientMapListResponseMessage

- “responseCode”: ResponseCode
- “clientMapNames”: array of ClientMapName

9.3 Client Localization (Module: *ClientLocalization*)

9.3.1 clientLocalizationStart

Requests to start self-localizing within the map.

- QueryType: ClientLocalizationQueryMessage
- ResponseType: ClientLocalizationResponseMessage

Query Value

- sessionId: the session requesting the mode change

Response Value

- response code as defined in the relevant section

9.3.2 clientLocalizationStop

Requests to stop self-localizing.

- QueryType: ClientLocalizationQueryMessage
- ResponseType: ClientLocalizationResponseMessage

Query Value

- sessionId: the session requesting the mode change

Response Value

- response code as defined in the relevant section

9.3.3 clientLocalizationSetSeed

Provides the Localization Client with an externally-derived pose estimate. This estimate can assist the client in self-localizing if the current pose is unknown. The current pose may be unknown if the client has just started up or if the localization was lost. If a seed is set while the system is not receiving any laser data, this seed will be used once laser data is once again received by the Localization Client.

- QueryType: ClientLocalizationSeedMessage
- ResponseType: ClientLocalizationResponseMessage

Query Value

- sessionId: the session providing the pose estimate
- enforceSeed : If the pose is enforced, the client will only consider this pose estimate when re-localizing. In return, the client will discard all internally-generated estimates of the pose. Note that the Localization Client may, however, generate additional pose estimates afterwards.
- seedPose : the pose that the Localization Client should use to aid in localization

Response Value

- response code as defined in the relevant section

9.3.4 Messages

ClientLocalizationQueryMessage

- “sessionId”: SessionId

ClientLocalizationResponseMessage

- “responseCode”: ResponseCode

ClientLocalizationSeedMessage

- “sessionId”: SessionId
- “enforceSeed”: boolean
- “seedPose”: Pose2D

9.4 Manual Map Alignment (Module: *ClientManualAlign*)

9.4.1 clientManualAlignStart

Requests to enter manual map alignment mode.

- QueryType: ClientManualAlignQueryMessage
- ResponseType: ClientManualAlignResponseMessage

Query Value

- sessionId: the session requesting the mode change

Response Value

- response code as defined in the relevant section

9.4.2 clientManualAlignStop

Requests to leave manual map alignment mode.

- QueryType: ClientManualAlignQueryMessage
- ResponseType: ClientManualAlignResponseMessage

Query Value

- sessionId: the session requesting the mode change

Response Value

- response code as defined in the relevant section

9.4.3 clientManualAlignGetPointCloud

Retrieves a point cloud representation of a map to assist in manual map alignment.

- QueryType: ClientManualAlignMapLevelMessage
- ResponseType: ClientManualAlignPointCloud2DResponseMessage

Query Value

- sessionId
- local map name for which the point cloud is requested
- level of the map from which the point cloud is generated (currently not used)

Response Value

- response code as defined in the relevant section
- point cloud of the requested map

9.4.4 clientManualAlignSet

Manually aligns a map, transforming it into the given reference frame. The aligned map is stored under a new name, which is returned in the response to this API method.

Note that the distance by which a map can be shifted through this method is restricted. This limitation is required to maintain the precision of the map when using a floating-point representation. In particular, maps may not be shifted by absolute values of more than 10,000 meters along each axis. Additionally, the user should ensure that the alignment does not result in a map with coordinates that exceed 100,000 meters along either axis. This issue can arise if a user performs repeated calls to clientManualAlignSet that shift a given map by a very large distance.

- QueryType: ClientManualAlignTransformMessage
- ResponseType: ClientManualAlignMapNameResponseMessage

Query Value

- sessionId
- transformation of the map from the old to the new reference frame
- name of the map to transform

Response Value

- response code as defined in the relevant section
- the name of the aligned map that was created by applying the transformation or an empty string on error

9.4.5 Messages

ClientManualAlignQueryMessage

- “sessionId”: SessionId

ClientManualAlignResponseMessage

- “responseCode”: ResponseCode

ClientManualAlignMapNameResponseMessage

- “responseCode”: ResponseCode
- “clientMapName”: ClientMapName or EmptyString

ClientManualAlignMapLevelMessage

- “sessionId”: SessionId
- “clientMapName” : ClientMapName
- “mapLevel” : Integer64

ClientManualAlignPointCloud2DResponseMessage

- “responseCode”: ResponseCode
- “pointCloud”: Container of media type application/PointCloud2D. Note: If the response code indicates an error, the container may be empty. In this case, the point cloud size will be set to 0.
- “pointCloudSize”: Size

ClientManualAlignTransformMessage

- “sessionId”: SessionId
- “NEWalignOLD” : ClientManualAlignOriginTransform2D
- “clientMapName” : ClientMapName

9.4.6 Objects

9.4.7 ClientManualAlignOriginTransform2D

- “x”: IEEE754Double within the range of $[-10^4, 10^4]$
- “y”: IEEE754Double within the range of $[-10^4, 10^4]$
- “a”: IEEE754Double

Describes a relative transformation between two 2D coordinate systems, given by the translation “x”, “y” (in meters) and the rotation angle “a” (in radians).

9.5 Global Alignment (Module: *ClientGlobalAlign*)

9.5.1 clientGlobalAlignAddObservation

Adds a new landmark observation to the recording currently in progress.

- QueryType: ClientGlobalAlignObservationMessage
- ResponseType: ClientGlobalAlignObservationIdResponseMessage

Query Value

- sessionId: The session adding the observation
- observation: The observation to add to the current recording

Response Value

- response code as defined in the relevant section
- an ID of the newly-added observation, which is unique within the current recording

9.5.2 clientGlobalAlignReplaceObservations

Replaces existing landmark observations within a saved recording. Note that the original recording is not modified: Instead, the method creates a copy of the recording where the specified observations have been replaced. The name of this newly modified recording is returned as part of the response value. This method may not be called while the Client is in the REC or VISUALRECORDING control modes.

- QueryType: ClientGlobalAlignReplaceObservationsMessage
- ResponseType: ClientGlobalAlignRecordingNameResponseMessage

Query Value

- the session replacing the observations
- the name of the recording within which to replace the existing observations
- array containing the unique IDs of the specific observations that should be replaced, as well as the new observations which will replace them

Response Value

- response code as defined in the relevant section
- The name of a new recording, or an empty string if an error occured. This recording is a copy of the recording from the query where the existing observations have been replaced

9.5.3 clientGlobalAlignDeleteObservations

Permanently deletes an existing landmark observation from a saved recording. Note that the original recording is not modified: Instead, the method creates a copy of the recording where the specified observations have been deleted. This method may not be called while the Client is in the REC or VISUALRECORDING control modes.

It may be desirable to temporarily disable a landmark observation instead of permantly deleting it. To do so, set the observation's type to `IGNORE` by calling clientGlobalAlignReplaceObservations.

- QueryType: ClientGlobalAlignObservationIdsMessage
- ResponseType: ClientGlobalAlignRecordingNameResponseMessage

Query Value

- the session deleting the observations
- the name of the recording from which to delete the existing observations
- the unique IDs of the specific observations that should be deleted

Response Value

- response code as defined in the relevant section
- The name of a new recording, or an empty string if an error occured. This recording is a copy of the recording from the query where the specified observations have been deleted

9.5.4 clientGlobalAlignListObservations

Retrieve the list of landmark observations which were added to an existing recording

- QueryType: ClientGlobalAlignRecordingNameMessage
- ResponseType: ClientGlobalAlignIdentifiedObservationArrayResponseMessage

Query Value

- the session requesting the list of observations
- the name of the recording for which to retrieve the observations

Response Value

- response code as defined in the relevant section
- list of all landmark observations and their unique ids within the given recording

9.5.5 Messages

ClientGlobalAlignObservationMessage

- “sessionId”: SessionId
- “observation”: ClientGlobalAlignObservation

ClientGlobalAlignReplaceObservationsMessage

- “sessionId”: SessionId
- “recordingName”: RecordingName
- “replacementObservations”: array of ClientGlobalAlignIdentifiedObservation

ClientGlobalAlignObservationIdsMessage

- “sessionId”: SessionId
- “recordingName”: RecordingName
- “observationIds”: a non empty array of unique ClientGlobalAlignObservationId

ClientGlobalAlignRecordingNameMessage

- “sessionId”: SessionId
- “recordingName”: RecordingName

ClientGlobalAlignRecordingNameResponseMessage

- “responseCode”: ResponseCode
- “recordingName”: RecordingName or EmptyString

ClientGlobalAlignObservationIdResponseMessage

- “responseCode”: ResponseCode
- “observationId”: ClientGlobalAlignObservationId

ClientGlobalAlignIdentifiedObservationArrayResponseMessage

- “responseCode”: ResponseCode
- “observations”: array of ClientGlobalAlignIdentifiedObservation

9.5.6 Objects

ClientGlobalAlignLandmarkName(AsciiCharacterString) Landmarks are referenced by non-empty strings of up to 240 characters in length.

ClientGlobalAlignSensorName (AsciiCharacterString) Sensors are referenced by non-empty strings of up to 240 characters in length.

ClientGlobalAlignObservation

- landmarkName: unique ClientGlobalAlignLandmarkName identifier of the observed landmark
- sensorName: unique ClientGlobalAlignSensorName identifier for the sensor that made the observation. Observations made by the same sensor need to have the same LASERstaticCalibSENSOR (see below).
- LASERstaticCalibSENSOR: ClientGlobalAlignSensorTransform2D - Transformation between the platform coordinate system and the coordinate system of the sensor that is perceiving the landmark
- SENSORlandmark: ClientGlobalAlignLandmarkPose2D - Observed pose of the landmark in the sensor coordinate system
- MAPlandmark: ClientGlobalAlignLandmarkPose2D - Precise pose of the landmark in the map coordinate system
- observationType: ClientGlobalAlignObservationType
- calibrationType: ClientGlobalAlignCalibrationType
- hasOrientation: boolean, indicates whether the orientation of the landmark is known. If this is set to false, the orientation angle of the landmark is ignored for this observation.

ClientGlobalAlignIdentifiedObservation

- “observationId”: ClientGlobalAlignObservationId
- “observation”: ClientGlobalAlignObservation

9.5.7 ClientGlobalAlignSensorTransform2D

- “x”: IEEE754Double within the range of $[-10^4, 10^4]$
- “y”: IEEE754Double within the range of $[-10^4, 10^4]$
- “a”: IEEE754Double

Describes a relative transformation between two 2D sensor coordinate systems, given by the translation “x”, “y” and the rotation angle “a”.

9.5.8 ClientGlobalAlignLandmarkPose2D

- “x”: IEEE754Double within the range of $[-10^4, 10^4]$
- “y”: IEEE754Double within the range of $[-10^4, 10^4]$
- “a”: IEEE754Double

Describes the pose of a landmark within a 2D coordinate system, given by the position “x”, “y” and the orientation angle “a”.

9.5.9 Types

ClientGlobalAlignObservationId (Integer64) Observation IDs encoded as integers.

ClientGlobalAlignObservationType (integer)

Label	Value	Description
ANNOTATION	1	Observation will not be used for global alignment. However, its pose will be optimized during map construction and can be used to visualize the pose or position of a landmark. This type must be unique for each landmark
REFERENCE	2	Observation is used for global alignment. Observations of this type should be precise
IGNORE	3	All observations of this type are ignored

ClientGlobalAlignCalibrationType (integer)

Label	Value	Description
FIX_ALL	0	Calibration is fixed for orientation and translation
OPTIMIZE_ALL	1	Calibration is optimized for orientation and translation
OPTIMIZE_TRANSLATION_-FIX_ORIENTATION	2	Calibration is fixed in orientation but optimized in translation
FIX_TRANSLATION_OPTI-MIZE_ORIENTATION	3	Calibration is optimized in orientation but fixed in translation

9.6 Laser Masking (Module: *ClientLaserMask*)

This functionality is only available while the Localization Client is in the MASK control mode.

9.6.1 clientLaserMaskingStart

Requests to enter the control mode MASK.

- QueryType: ClientLaserMaskQueryMessage
- ResponseType: ClientLaserMaskResponseMessage

Query Value

- sessionId: the session requesting the mode change

Response Value

- response code as defined in the relevant section

9.6.2 clientLaserMaskingStop

Requests to leave the control mode MASK.

- QueryType: ClientLaserMaskQueryMessage
- ResponseType: ClientLaserMaskResponseMessage

Query Value

- sessionId: the session requesting the mode change

Response Value

- response code as defined in the relevant section

9.6.3 clientLaserMaskingGetScan

Retrieves the next laser scan available to the Localization Client.

- QueryType: ClientLaserMaskQueryMessage
- ResponseType: ClientLaserMaskLaserScanResponseMessage

Query Value

- sessionId

Response Value

- response code as defined in the relevant section
- the next available laser scan

9.6.4 Messages

ClientLaserMaskQueryMessage

- “sessionId”: SessionId

ClientLaserMaskResponseMessage

- “responseCode”: ResponseCode

MaskLaserScanResponseMessage

- “responseCode”: ResponseCode
- “ranges”: array of IEEE754Double
- “intensities”: array of IEEE754Double
- “numBeams”: Size
- “angleInc”: IEEE754Double
- “minIntensity”: IEEE754Double
- “maxIntensity”: IEEE754Double
- “timeStart”: Timestamp
- “durationBeam”: TimeInterval
- “durationScan”: TimeInterval
- “durationRotation”: TimeInterval

- “scanNum”: Size
- “hasIntensities”: boolean
- “angleStart”: IEEE754Double
- “angleEnd”: IEEE754Double
- “minRange”: IEEE754Double
- “maxRange”: IEEE754Double

9.7 User Account Management (Module: *ClientUser*)

9.7.1 Types

ClientUserGroup An AsciiCharacterString containing one of the following values: admin, user, observer.

9.7.2 clientUserAdd

Adds a new user account to the Localization Client.

- QueryType: ClientUserAddMessage
- ResponseType: ClientUserResponseMessage

Query Value

- sessionId: the session adding the new account
- userName: the name of the new user which should be added and must be unique in the system among the different user groups
- userPassword: the password of the new user
- userGroups: the user groups to which the new user belongs (must contain exactly one group)

Response Value

- response code as defined in the relevant section

9.7.3 clientUserDelete

Deletes a user account from the Localization Client. Only user of admin group can delete users except himself.

- QueryType: ClientUserDeleteMessage
- ResponseType: ClientUserResponseMessage

Query Value

- sessionId: the session deleting the user
- userName: the name of the user which should be deleted

Response Value

- response code as defined in the relevant section

9.7.4 clientUserChangePassword

Changes a user's password stored in the Localization Client.

- QueryType: ClientUserPasswordChangeMessage
- ResponseType: ClientUserResponseMessage

Query Value

- sessionId: the session requesting the password change
- userName: the name of the user whose password should be changed
- newUserPassword: the new password

Response Value

- response code as defined in the relevant section

9.7.5 clientUserChangeOwnPassword

Changes a user's own password stored in the Localization Client. Unlike clientUserChangePassword, this method requires the user's current password to be sent. In return, this method is not restricted to the admin group, and can be used by all users to change their own passwords.

- QueryType: ClientUserOwnPasswordChangeMessage
- ResponseType: ClientUserResponseMessage

Query Value

- sessionId: the session requesting the password change
- newUserPassword: the new password
- oldUserPassword: the current password

Response Value

- response code as defined in the relevant section

9.7.6 clientUserList

Retrieves the list of all user accounts known to the Localization Client.

- QueryType: ClientUserQueryMessage
- ResponseType: ClientUserListResponseMessage

Query Value

- sessionId: the session requesting the user list

Response Value

- response code as defined in the relevant section
- array of user info as in ClientUserInfo

9.7.7 clientUserListGroup

Retrieves the list of all user groups known to the Localization Client.

- QueryType: ClientUserQueryMessage
- ResponseType: ClientUserGroupListResponseMessage

Query Value

- sessionId: the session requesting the user group list

Response Value

- response code as defined in the relevant section
- array of user groups known to the Localization Client

9.7.8 Messages

ClientUserAddMessage

- sessionId: SessionId
- userName: AsciiCharacterString
- userPassword: AsciiCharacterString
- userGroups: array of ClientUserGroup

ClientUserDeleteMessage

- sessionId: SessionId
- userName: AsciiCharacterString

ClientUserPasswordChangeMessage

- sessionId: SessionId
- userName: AsciiCharacterString
- newUserPassword: AsciiCharacterString

ClientUserOwnPasswordChangeMessage

- sessionId: SessionId
- newUserPassword: AsciiCharacterString
- oldUserPassword: AsciiCharacterString

ClientUserResponseMessage

- responseCode: ResponseCode

ClientUserQueryMessage

- sessionId: SessionId

ClientUserListResponseMessage

- responseCode: ResponseCode
- userInfoList: array of ClientUserInfo

ClientUserGroupListResponseMessage

- responseCode: ResponseCode
- userGroups: array of AsciiCharacterString

9.7.9 Objects

ClientUserInfo

- “userName”: AsciiCharacterString
- “userGroups”: array of ClientUserGroup

Contains information about a user; specifically, the user name and a list of user groups to which the user belongs. in current version the array should have the size one, since the user can only belong to one group.

10 Map Server – RPC Methods

The main task of the Map Server is to store, update, and distribute the maps used by the Laser Localization Clients.

Each map consists of multiple layers used either for localization, management or visualization of the map. Currently, these following layers are used:

- laser-Map: Built from laser scans of the environment

In future releases, maps may include other layers. Such layers may be generated from different sensors or contain additional information. Each layers is uniquly identified by a fingerprint and contains the timestamps “createdAt”, “updatedAt”, “sentAt” and “receivedAt”.

Additionally, a map includes the ‘level’ metadata, which is not used at the moment.

In this chapter, a map refers to any map stored by the Map Server. Such maps have to be built using the Localization Client’s map building process.

10.1 Server-side Map Management (Module: *ServerMap*)

10.1.1 serverMapGetInfo

Returns all information about a given map

- QueryType: ServerMapNameMessage
- ResponseType: ServerMapInfoResponseMessage

Query Value

- sessionId: the session requesting the map information
- the name of the map for which the information is requested

Response Value

- response code as defined in the relevant section
- level: currently not used
- loaded flag: set if the map is loaded into the Map Server's memory
- type: currently not used
- "sentAt": the time the map was last sent by the Localization Client to the Map Server
- "receivedAt": the time the map was originally received by the Map Server from the Localization Client
- "updatedAt": the timestamp of the last laser scan that was used to update the map (not set if the map was never updated)
- "createdAt": the timestamp of the last laser scan included when creating the map

10.1.2 serverMapList

Requests a list of all maps mananged by the map server.

- QueryType: ServerMapQueryMessage
- ResponseType: ServerMapListResponseMessage

Query Value

- sessionId: the session requesting the list

Response Value

- response code as defined in the relevant section
- list of maps known to the map server

10.1.3 serverMapRename

Renames a map managed by the Map Server.

- QueryType: ServerMapRenameMessage
- ResponseType: ServerMapListResponseMessage

Query Value

- sessionId: the session requesting the map rename
- current name of the server-side map
- desired new name of the server-side map

Response Value

- response code as defined in the relevant section

10.1.4 serverMapDelete

Deletes a map managed by the Map Server.

- QueryType: ServerMapNameMessage
- ResponseType: ServerMapListResponseMessage

Query Value

- sessionId: the session requesting the map deletion
- name of the server-side map

Response Value

- response code as defined in the relevant section

10.1.5 serverMapGetPointCloud

Retrieves a point cloud representation of a map managed by the server.

- QueryType: ServerMapNameMessage
- ResponseType: ServerMapPointCloud2DResponseMessage

Query Value

- sessionId: the session requesting the point cloud
- map name for which the point cloud is requested

Response Value

- response code as defined in the relevant section
- point cloud of the requested map
- size of the point cloud

10.1.6 serverMapGetImage

Retrieves a raster image of the map with a given image size.

- QueryType: ServerMapImageSizeMessage
- ResponseType: ServerMapPngImageResponseMessage

Query Value

- sessionId: the session requesting the image
- width in pixels: integer in range from [1, 20000]
- height in pixels: integer in range from [1, 20000]
- level of the map (not used)
- name of the server-side map for which the image is requested

Response Value

- response code as defined in the relevant section
- pose of the image origin within the map
- width in meter
- height in meter
- container holding the requested image in PNG format

10.1.7 serverMapGetImageWithResolution

Retrieves a raster image of the map with a given image resolution.

- QueryType: ServerMapImageResolutionMessage
- ResponseType: ServerMapPngImageResponseMessage

Query Value

- sessionId: the session requesting the image
- resolution: This value holds the image resolution in pixels per meter (must be greater than zero). The chosen resolution must not result in an image with a width or height greater than 20000 pixels. Choosing such a large resolution will result in an error.
- name of the server-side map for which the image is requested

Response Value

- response code as defined in the relevant section
- pose of the image origin within the map
- width in meter
- height in meter
- container holding the requested image in PNG format

10.1.8 serverMapGetMapThumbnail

Retrieves a thumbnail-sized raster image of the map.

- QueryType: ServerMapLevelMessage
- ResponseType: ServerMapPngImageResponseMessage

Query Value

- sessionId: the session requesting the image
- level of the map (not used)
- name of the server-side map for which the thumbnail is requested

Response Value

- response code as defined in the relevant section
- pose of the image origin within the map
- width in meter
- height in meter
- container holding the requested image in PNG format

10.1.9 Messages

ServerMapQueryMessage

- sessionId: SessionId

ServerMapNameMessage

- “sessionId”: SessionId
- “serverMapName”: ServerMapName

ServerMapLevelMessage

- “sessionId”: SessionId
- “serverMapName”: ServerMapName
- “mapLevel”: Integer64

ServerMapRenameMessage

- “sessionId”: SessionId
- “currentServerMapName”: ServerMapName
- “desiredServerMapName”: ServerMapName

ServerMapListResponseMessage

- “responseCode”: ResponseCode
- “serverMapNames”: array of ServerMapName

ServerMapInfoResponseMessage

- “responseCode”: ResponseCode
- “level”: Integer64
- “loaded”: boolean
- “type”: Integer64
- “fingerprint”: Integer64
- “createdAt”: Timestamp
- “updatedAt”: Timestamp
- “sentAt”: Timestamp
- “receivedAt”: Timestamp

ServerMapPointCloud2DResponseMessage

- “responseCode”: ResponseCode
- “pointCloud”: Container of media type application/PointCloud2D.
- “pointCloudSize”: Size

ServerMapImageSizeMessage

- “sessionId”: SessionId
- “width”: Size
- “height”: Size
- “level”: Integer64
- “serverMapName”: ServerMapName

ServerMapImageResolutionMessage

- “sessionId”: SessionId
- “resolution”: IEEE754Double, greater than 0
- “serverMapName”: ServerMapName

ServerMapPngImageResponseMessage

- “responseCode”: ResponseCode
- “MAPimageOrigin”: Pose2D; pose of the image origin within the map; the image origin is the lower left corner of the image; the x-axis points to the lower right corner and the y-axis points to the upper left corner.
- “width”: IEEE754Double, greater than or equal to 0; width of the map in the Cartesian coordinates system (m).
- “height”: IEEE754Double, greater than or equal to 0; height of the map in the Cartesian coordinates system (m).
- “image”: Container of media type image/png. Note: If the response code indicates an error, the content of the container may be empty, and cannot be interpreted as a PNG image.

10.2 Server-side Internal Settings (Module: *ServerInternal*)

The methods within this module are currently for internal use only, and should not be called directly. However, they are documented here for the sake of completeness.

10.2.1 serverInternalFleetConfigSet

Set the fleet management configuration (internal use only)

- QueryType: ServerInternalFleetConfigSetingsMessage
- ResponseType: ServerInternalResponseMessage

Query Value

- sessionId: the session setting the configuration
- the new fleet management settings

Response Value

- response code as defined in the relevant section

10.2.2 serverInternalFleetConfigGet

Get the fleet management configuration (internal use only)

- QueryType: ServerInternalQueryMessage
- ResponseType: ServerInternalFleetConfigSetingsResponseMessage

Query Value

- sessionId: the session retrieving the configuration

Response Value

- response code as defined in the relevant section
- the stored fleet management settings

10.2.3 Messages

ServerInternalQueryMessage

- “sessionId”: SessionId

ServerInternalResponseMessage

- “responseCode”: ResponseCode

ServerInternalFleetConfigSettingsMessage

- “sessionId”: SessionId
- “settings”: ServerInternalFleetConfigSettings

ServerInternalFleetConfigSettingsResponseMessage

- “responseCode”: ResponseCode
- “settings”: ServerInternalFleetConfigSettings

10.2.4 Objects

ServerInternalFleetConfigSettings A string with a length of between 1 and 10000 characters.

10.3 User Management (Module: *ServerUser*)

10.3.1 Types

ServerUserGroup An AsciiCharacterString containing one of the following values: admin, user, observer.

10.3.2 serverUserAdd

Adds a new user account to the Map Server.

- QueryType: ServerUserAddMessage
- ResponseType: ServerUserResponseMessage

Query Value

- sessionId: the session requesting the information
- userName: the name of the new user which will be added
- userPassword: the password of the new user
- userGroup: the group to which the new user belongs

Response Value

- response code as defined in the relevant section

10.3.3 serverUserDelete

Deletes a user account from the Localization Client. Only user of admin group can delete users except himself.

- QueryType: ServerUserDeleteMessage
- ResponseType: ServerUserResponseMessage

Query Value

- sessionId: the session requesting the information
- userName: the name of the user which will be deleted

Response Value

- response code as defined in the relevant section
- responseId as userName in ServerUserAddMessage for associating the response

10.3.4 serverUserChangePassword

Changes a user's password stored in the Map Server.

- QueryType: ServerUserPasswordChangeMessage
- ResponseType: ServerUserResponseMessage

Query Value

- sessionId: the session requesting the information
- userName: the name of the user whose password will be changed
- newUserPassword: the new password

Response Value

- response code as defined in the relevant section
- responseId as userName in ServerUserAddMessage for associating the response

10.3.5 serverUserChangeOwnPassword

Changes a user's own password stored in the Map Server. Unlike serverUserChangePassword, this method requires the user's current password to be sent. In return, this method is not restricted to the admin group, and can be used by all users to change their own passwords.

- QueryType: ServerUserOwnPasswordChangeMessage
- ResponseType: ServerUserResponseMessage

Query Value

- sessionId: the session requesting the information
- newUserPassword: the new password
- oldUserPassword: the current password

Response Value

- response code as defined in the relevant section
- responseId as userName in ServerUserAddMessage for associating the response

10.3.6 serverUserList

Retrieves the list of all user accounts known to the Map Server.

- QueryType: ServerUserQueryMessage
- ResponseType: ServerUserListResponseMessage

Query Value

- sessionId: the session requesting the information

Response Value

- response code as defined in the relevant section
- array of user info as in ServerUserInfo

10.3.7 serverUserListGroup

Retrieves the list of all user groups known to the Map Server.

- QueryType: ServerUserQueryMessage
- ResponseType: ServerUserGroupListResponseMessage

Query Value

- sessionId: the session requesting the information

Response Value

- response code as defined in the relevant section
- array of user groups known to the MapServer

10.3.8 Messages

ServerUserAddMessage

- sessionId: SessionId
- userName: AsciiCharacterString
- userPassword: AsciiCharacterString
- userGroups: array of ServerUserGroup

ServerUserDeleteMessage

- sessionId: SessionId
- userName: AsciiCharacterString

ServerUserPasswordChangeMessage

- sessionId: SessionId
- userName: AsciiCharacterString
- newServerUserPassword: AsciiCharacterString

ServerUserOwnPasswordChangeMessage

- sessionId: SessionId
- newServerUserPassword: AsciiCharacterString
- oldServerUserPassword: AsciiCharacterString

ServerUserResponseMessage

- responseCode: ResponseCode
- responseId: AsciiCharacterString

ServerUserQueryMessage

- sessionId: SessionId

ServerUserListResponseMessage

- responseCode: ResponseCode
- userInfoList: array of ServerUserInfo

ServerUserGroupListResponseMessage

- responseCode: ResponseCode
- userGroups: array of AsciiCharacterString

10.3.9 Objects

ServerUserInfo

- “userName”: AsciiCharacterString; user name
- “userGroups”: array of ServerUserGroup; a list of user groups to which the user belongs (currently user can only belong to one group)

11 Support – RPC Methods

11.1 Support Reports (Module: *SupportReport*)

11.1.1 supportReportCreate

Creates a new support report that documents the current system state. The report is added to a report list, from which it can be retrieved later.

- QueryType: SupportReportQueryMessage
- ResponseType: SupportReportNameResponseMessage

Query Value

- sessionId: the session requesting the report

Response Value

- response code as defined in the relevant section
- name of the newly created support report (empty string on error)

11.1.2 supportReportCreateMinimal

Creates a new minimal support report that provides a limited documentation of the current system state. Note that while this minimal report is much smaller, it may also lack some of the relevant information. The report is added to a report list, from which it can be retrieved later.

- QueryType: SupportReportQueryMessage
- ResponseType: SupportReportNameResponseMessage

Query Value

- sessionId: the session requesting the minimal report

Response Value

- response code as defined in the relevant section
- name of the newly created minimal support report (empty string on error)

11.1.3 supportReportSetDescription

Stores a support report description, which can include additional support information supplied by a user. This description will be stored in a buffer and included in the next support reports created under this session. Calling this method repeatedly without creating a report will overwrite the previous description buffer.

- QueryType: SupportReportDescriptionMessage
- ResponseType: SupportReportResponseMessage

Query Value

- sessionId: the session for which the description should be set
- reportDescription: the text which should be added to the next support report

Response Value

- response code as defined in the relevant section

11.1.4 supportReportList

Retrieves a list of all known support reports.

- QueryType: SupportReportQueryMessage
- ResponseType: SupportReportListResponseMessage

Query Value

- sessionId: the session requesting the list

Response Value

- response code as defined in the relevant section
- the names of all known support reports

11.1.5 supportReportGetPath

Retrieves the specified support report.

- QueryType: SupportReportNameMessage
- ResponseType: SupportReportNameUrlResponseMessage

Query Value

- sessionId: the session requesting the report
- the name of the report being requested

Response Value

- response code as defined in the relevant section
- a url including a authorization token, empty string on error

The url is the ressource from which the support report can be retrieved including a token authorizing the caller to download this resource doing a url “get” request matching the pattern of the form:

```
<PROTOCOL>://<SUPPORT_REPORT_DOMAIN>:<SUPPORT_REPORT_PORT>/<URL>
```

e.g.

```
http://myhost:8084/report-2019-07-19T11:56:39.476182Z.tar?token=01234567-1234-4012-b012-01234567890a
```

11.1.6 supportReportDelete

Deletes the specified support report.

- QueryType: SupportReportNameMessage
- ResponseType: SupportReportListResponseMessage

Query Value

- sessionId: the session requesting the operation
- the name of the report to be deleted

Response Value

- response code as defined in the relevant section
- list of reports after the query has been processed successfully, empty on error

11.1.7 Messages

SupportReportQueryMessage

- “sessionId”: SessionId

SupportReportNameResponseMessage

- “responseCode”: ResponseCode
- “reportName”: SupportReportName or EmptyString

SupportReportDescriptionMessage

- “sessionId”: SessionId
- “reportDescription”: string

SupportReportListResponseMessage

- “responseCode”: ResponseCode
- “reportNames”: array of SupportReportName

SupportReportNameMessage

- “sessionId”: SessionId
- “reportName”: SupportReportName

SupportReportResponseMessage

- “responseCode”: ResponseCode

SupportReportNameUrlResponseMessage

- “responseCode”: ResponseCode
- “reportNameUrl”: SupportReportNameUrl or EmptyString

11.1.8 Objects

SupportReportName (AsciiCharacterString, length > 0) Support reports are referenced by non-empty strings of up to 240 characters in length. These names may not contain forward slashes (‘/’).

SupportReportNameUrl (AsciiCharacterString, length > 0) Support reports URL are referenced by non-empty strings of up to 240 characters in length. These names may not contain forward slashes (‘/’).

11.2 System Backup, Update, Recovery (Module: *SupportRecovery*)

11.2.1 supportRecoveryList

Retrieves a list of all available recovery points.

- QueryType: SupportRecoveryQueryMessage
- ResponseType: SupportRecoveryListResponseMessage

Query Value

- sessionId: the session requesting the information

Response Value

- response code as defined in this section
- the names of all available recovery points

11.2.2 supportRecoveryCreate

Creates a new recovery point that provides a backup of the current system state. The recovery point is added to a recovery point list, from which it can be retrieved later. A recovery point contains encrypted data.

- QueryType: SupportRecoveryQueryMessage
- ResponseType: SupportRecoveryNameResponseMessage

Query Value

- sessionId: the session requesting the creation of the recovery point

Response Value

- response code as defined in this section
- name of the newly created recovery point (empty string on error)

11.2.3 supportRecoveryDelete

Deletes the specified recovery point.

- QueryType: SupportRecoveryNameMessage
- ResponseType: SupportRecoveryListResponseMessage

Query Value

- sessionId: the session requesting the operation
- the name of the recovery point to be deleted

Response Value

- response code as defined in this section
- list of recovery points after the query has been processed successfully, empty on error

11.2.4 supportRecoveryFactoryReset

Reset the system to the factory state with its original settings. Be cautious, this API call removes all the current settings and cannot be revoked.

- QueryType: SupportRecoveryQueryMessage
- ResponseType: SupportRecoveryResponseMessage

Query Value

- sessionId: the session requesting the operation

Response Value

- response code as defined in this section

11.2.5 supportRecoveryFrom

Recovers the state of the associated product element from a selected recovery point.

- QueryType: SupportRecoveryNameMessage
- ResponseType: SupportRecoveryResponseMessage

Query Value

- sessionId: the session requesting the operation
- the name of the recovery point

Response Value

- response code as defined in this section

11.2.6 Messages

SupportRecoveryQueryMessage

- “sessionId”: SessionId

SupportRecoveryNameMessage

- “sessionId”: SessionId
- “recoveryName”: SupportRecoveryName

SupportRecoveryResponseMessage

- “responseCode”: ResponseCode

SupportRecoveryNameResponseMessage

- “responseCode”: ResponseCode
- “recoveryName”: SupportRecoveryName or EmptyString

SupportRecoveryListResponseMessage

- “responseCode”: ResponseCode
- “recoveryNames”: array of SupportRecoveryName

11.2.7 Objects

SupportRecoveryName (AsciiCharacterString, length > 0) Recovery points are referenced by non-empty strings of up to 240 characters in length. These names may not contain forward slashes (‘/’).

12 Binary Interfaces

Binary Interfaces handle low-latency, high-throughput transmission of homogenous datagrams from and to the user. This includes transmitting the sensor pose for localization, transmitting of the current scan and map for visualization, and receiving user-supplied sensor data. Here, a Datagram refers to a packed (unaligned and unpadded) data structure. They carry the data transmitted by a Binary Interface; Datagrams may thus be regarded as the equivalent of Messages in the RPC Interface. The user can access Binary Interfaces provided by the product elements through TCP/IP. Refer to the approproate section for a list of all Binary Interfaces and their default TCP ports. Note that the Binary Interfaces use little-endian byte order when transmitting integers as part of datagrams.

Just like the RPC Interface, all Binary Interfaces and their associated data types belong to a specific module. Additionally, some common datatypes are used across multiple Binary Interfaces.

12.1 Common Definitions (Shared across Modules)

12.1.1 Common Types

The following types are shared across multiple Binary Interfaces. They are never transmitted on their own, but only as parts of other datagrams.

IEEE754Single A floating point number in IEEE754single-precision format.

IEEE754Double A floating point number in IEEE754double-precision format.

Boolean A little-endian, unsigned integer of 8 bits with a possible value within the interval [0, 1].

UnsignedInteger64 A little-endian, unsigned integer of 64 bits with a possible value within the interval [0, 2^{64}) or [0, 18446744073709551615].

SignedInteger64 A little-endian, two-complement signed integer of 64 bits with a possible value within the interval [-2^{63}, 2^{63}) or [-9223372036854775808, 9223372036854775807].

UnsignedInteger32 An little-endian, unsigned integer of 32 bits with a possible value within the interval [0, 2^{32}) or [0, 4294967295].

SignedInteger32 A little-endian, two-complement signed integer of 32 bits with a possible value within the interval [-2^{31}, 2^{31}) or [-2147483648, 2147483647].

PrintableAsciiCharacter A little-endian, unsigned integer of 8 bits with a possible value within the interval [$0x20$, $0x7e$]. This corresponds to those characters within the ASCII standard that can be printed directly, such as a-z, A-Z or 0-9. Note that all such characters are also valid and printable characters in UTF-8 encoding and can be interpreted accordingly.

Pose2DDouble A 2D pose represented by Cartesian x/y coordinates and a yaw angle.

Name	Size (B)	Type	Restriction
x	8	IEEE754Double	No restrictions.
y	8	IEEE754Double	No restrictions.
yaw	8	IEEE754Double	in [$-\pi_d$, π_d]

Pose2DSingle A 2D pose represented by Cartesian x/y coordinates and a yaw angle. Single precision format.

Name	Size (B)	Type	Restriction
x	4	IEEE754Single	No restrictions.
y	4	IEEE754Single	No restrictions.
yaw	4	IEEE754Single	in $[-\pi_f, \pi_f]$

Position2DDouble A 2D position represented by Cartesian x/y coordinates.

Name	Size (B)	Type	Restriction
x	8	IEEE754Double	No restrictions.
y	8	IEEE754Double	No restrictions.

Position2DSingle A 2D position represented by Cartesian x/y coordinates.

Name	Size (B)	Type	Restriction
x	4	IEEE754Single	No restrictions.
y	4	IEEE754Single	No restrictions.

12.1.2 Common Constants

The irrational number π cannot be accurately represented using IEEE754 floating point numbers. When specifying restrictions for angles, we therefore replace π with the approximations π_f and π_d. Here, π_f is the most accurate possible IEEE754 single-precision approximation of π. By definition, π_f is thus the IEEE754 single-precision number that is closest to π and thus minimizes $|\pi - \pi_f|$. Similarly, π_d is the double-precision representation of π, and thus the IEEE754 double-precision number that minimizes $|\pi - \pi_d|$.

12.1.3 Arrays

Definition An array is a data structure that consists of any number of objects of the same type. The number of such elements within a given array is called the array's

length. Note that the length of an array may not be known beforehand. Consequently, it must always be transmitted *before* the actual objects that make up the array.

Datagram schema Note the following example:

Name	Size (B)	Type	Restriction
name of array		Array	
–>length	4	UnsignedInteger32	<= 134217728, >= 0
–>elements	8	Position2DSingle	|x|,|y| <= 1.e+05

This is read as follows:

- first 4 bytes are the `length` transmitted as an UnsignedInteger32
- next 8 bytes are the first element as an Position2DSingle
- next 8 bytes are the second element as an Position2DSingle
- ... repeat until a total of `length` elements have been transmitted

12.1.4 Common Datagrams

The following datagrams are shared between different modules.

MapDatagram A map represented as a point cloud. Each point corresponds to a single surface point of an obstacle, as detected by a laser scanner.

Name	Size (B)	Type	Restriction
map		Array	
–>length	4	UnsignedInteger32	<= 134217728, >= 0
–>elements	8	Position2DSingle	|x|,|y| <= 1.e+05

12.2 Control Output (Module: *ClientControl*)

12.2.1 Interfaces

ClientControlMode Interface The ClientControlMode Binary Interface provides information about the current operating mode of the Localization Client. Specifically, it

regularly transmits the mode as defined in Client Control Mode, using the ClientControlModeDatagram.

12.2.2 Datagrams

ClientControlModeDatagram

Name	Size (B)	Type	Restriction
controlMode	4	UnsignedInteger32	

The client modes are expressed as a 32-bit bitfield.This bitfield is divided into 3-bit ClientStates. Each ClientState within the field expresses the state of a single client mode. These client modes correspond to those listed in ClientMode. The layout of the bitfield is as follows:

Bits	31-18	17-15	14-12	11-9	8-6	5-3	2-0
Mode	Unused	VISUALRECORDING	MAP	LOC	REC	ALIGN	MASK

Here, bit 0 is the least significant bit of the 32-bit bitfield, while bit 31 is the most significant bit.

ClientState These values are used to identify the state of each control modes within the Localizaton Client:

Label	Value (3 bit unsigned integer)	Description
INIT	0	Control mode is initializing and will be READY soon
READY	1	Control mode is inactive, but ready to be activated
RUN	2	Control mode is currently active
NOT_AVAILABLE	4	Control mode cannot be used

12.3 Localization Output (Module: *ClientLocalization*)

12.3.1 Interfaces

ClientLocalizationPose Interface The ClientLocalizationPose interface provides continuous absolute and relative 2D planar poses. It operates while the Localization Client is in localization mode and uses the ClientLocalizationPoseDatagram. This interface should remain stable across future versions. It therefore emits 3D poses (six degrees of freedoms) and unique IDs, even though these features are not yet supported.

ClientLocalizationVisualization Interface The ClientLocalizationVisualization interface provides information needed to visualize and evaluate the localization process. It operates while the Localization Client is in localization mode and uses the ClientLocalizationVisualizationDatagram.

ClientLocalizationMap Interface The ClientLocalizationMap provides the map that is currently used for self-localization. It operates while the Localization Client is in localization mode and uses the MapDatagram.

12.3.2 Datagrams

ClientLocalizationPoseDatagram

Name	Size (B)	Type	Restriction
age	8	IEEE754Double	No restrictions.
timestamp	8	IEEE754Double	<= 1.e+12, >= 0.e+00
uniqueId	8	UnsignedInteger64	No restrictions.
state	4	SignedInteger32	one of: -3, -2, -1, 1, 2, 3, 4,
poseX	8	IEEE754Double	<= 1.e+12, >= -1.e+12
poseY	8	IEEE754Double	<= 1.e+12, >= -1.e+12
poseYaw	8	IEEE754Double	in $[-\pi_d, \pi_d]$
covariance_1,1	8	IEEE754Double	>= 0.e+00
covariance_1,2	8	IEEE754Double	No restrictions.
covariance_1,3	8	IEEE754Double	No restrictions.
covariance_2,2	8	IEEE754Double	>= 0.e+00
covariance_2,3	8	IEEE754Double	No restrictions.

Name	Size (B)	Type	Restriction
covariance_3,3	8	IEEE754Double	>= 0.e+00
poseZ	8	IEEE754Double	<= 1.e+12, >= -1.e+12
quaternion_w	8	IEEE754Double	<= 1.e+00, >= -1.e+00
quaternion_x	8	IEEE754Double	<= 1.e+00, >= -1.e+00
quaternion_y	8	IEEE754Double	<= 1.e+00, >= -1.e+00
quaternion_z	8	IEEE754Double	<= 1.e+00, >= -1.e+00
epoch	8	UnsignedInteger64	No restrictions.
lidarOdoPoseX	8	IEEE754Double	<= 1.e+12, >= -1.e+12
lidarOdoPoseY	8	IEEE754Double	<= 1.e+12, >= -1.e+12
lidarOdoPoseYaw	8	IEEE754Double	in $[-\pi_d, \pi_d]$

- **age**: The time passed between receiving the most recent laser scan and the generation of this Datagram. Note that this field is calculated from the timestamp of the laser scan. Depending on the laser in question, this timestamp may be provided by the sensor itself. If the clocks of the laser sensor and the host system are not synchronized, this field may thus contain incorrect values, including negative ones. The user should thus make sure that the clocks are properly synchronized before using this value.
- **timestamp**: The time at which the Localization Client received the laser scan used to generate this Datagram.
- **uniqueId**: This field is equal to the uniqueId of the scan from which this datagram was calculated, as provided via the ClientSensorLaser interface. If the ClientSensorLaser interface is not used, this value is undefined.
- **state**: The localization status.
- **poseX**, **poseY**, **poseYaw**: The estimated Cartesian (x,y)-coordinates and yaw angle of the sensor, given in the map reference frame.
- **covariance_1,1**, **covariance_1,2**, **covariance_1,3**, **covariance_2,2**, **covariance_2,3**, **covariance_3,3**: The upper triangle of the covariance matrix of the pose, as estimated by the system. If the system cannot calculate the covariance matrix, the main diagonal of the matrix will be set to infinity, while all other values will be zero. The two numbers **n,m** of each value refer to the row **n** and column **m** of the entry within the matrix. As the covariance matrix is symmetrical, the lower triangle can simply be reconstructed from the upper triangle. Note that this covariance matrix is currently given in arbitrary, non-physical units. Consequently, users should only interpret these covariance matrices in comparison to each other.
- **poseZ**: this value is unused in this version of the API and will always be 0.
- **quaternion_w**, **quaternion_x**, **quaternion_y**, **quaternion_z**: The sensor orienta-

tion given as quaternions. Note that the Localization Client currently only emits orientations that lie within a single plane.

- **epoch**: Only Datagrams with the same epoch can be treated as locally precise and coherent. If the self-localization was temporarily disrupted, the system will indicate this by incrementing the epoch.
- **lidarOdoPoseX**, **lidarOdoPoseY**, **lidarOdoPoseYaw**: The estimated Cartesian (x,y)-coordinates and yaw angle of the sensor, given in an arbitrary, relative reference frame.

ClientLocalizationVisualizationDatagram

Name	Size (B)	Type	Restriction
timestamp	8	IEEE754Double	<= 1.e+12, >= 0.e+00
uniqueId	8	UnsignedInteger64	No restrictions.
locState	4	SignedInteger32	one of: -3, -2, -1, 1, 2, 3, 4,
poseX	8	IEEE754Double	<= 1.e+12, >= -1.e+12
poseY	8	IEEE754Double	<= 1.e+12, >= -1.e+12
poseYaw	8	IEEE754Double	in $[-\pi_d, \pi_d]$
delay	8	IEEE754Double	<= TYPE_MAX, >= 0.e+00
scan		Array	
–>length	4	UnsignedInteger32	<= 20000000, >= 0
–>elements	8	Position2DSingle	|x|,|y| <= 1.e+05

- **timestamp**: The time at which the Localization Client received the laser scan used to generate this Datagram.
- **uniqueId**: This field is equal to the uniqueId of the scan from which this datagram was calculated, as provided via the ClientSensorLaser interface. If the ClientSensorLaser interface is not used, this value is undefined.
- **locState**: The localization status.
- **poseX**, **poseY**, **poseYaw**: The estimated Cartesian (x,y)-coordinates and yaw angle of the sensor, given in the map reference frame.
- **delay** : The relative delay within the system when processing laser scans. 0 : no delay; 1.0 : delay causes scans to drop.
- **scan**: A 2D point cloud representing the most recent laser scan, given in the current map frame.

LocStatus

Label	Value (Signed-Integer32)	Description
INFO_LOCALIZATION_NOT_-RUNNING	-3	Laser Localization not RUNNING
NOT_LOCALIZED_STARTUP	-2	Not localized, still in start-up; May indicate that the Localization Client has not yet received a map from the Map Server
NOT_LOCALIZED	-1	Not localized, Localization Client has received a map and is attempting to self-localize
LOCALIZED_ODO_ONLY	1	Localization is available, but the most recent laser scans could not be matched to the map. Consequently, the accuracy and reliability of the localization may be degraded
LOCALIZED	2	Localized, laser scans are succesfully matched to the map
LOCALIZED_INTERNALUSE1	3	Localized, laser scans are succesfully matched to the map. For internal use only, should be considered equivalent to LOCALIZED
LOCALIZED_INTERNALUSE2	4	Localized, laser scans are succesfully matched to the map. For internal use only, should be considered equivalent to LOCALIZED

12.4 Visual Recording Output (Module: *ClientRecording*)

12.4.1 Interfaces

ClientRecordingVisualization Interface The ClientRecordingVisualization interface provides information needed to visualize and evaluate the current recording process.

It operates while the Localization Client is in recording mode and uses the ClientRecordingVisualizationDatagram. This Datagram contains the most recent laser scan in the map reference frame. It also contains the estimated path taken by the sensor, and the path as estimated without loop closures. Additionally, the Datagram contains possible loop closure candidates, as well as additional information.

ClientRecordingMap Interface The ClientRecordingMap provides the user with the current state of the map that could be generated using this recording. It operates while the Localization Client is in recording mode and uses the MapDatagram.

12.4.2 Datagrams

ClientRecordingVisualizationDatagram

Name	Size (B)	Type	Restriction
timestamp	8	IEEE754Double	<= 1.e+12, >= 0.e+00
visualizationId	8	UnsignedInteger64	No restrictions.
status	4	SignedInteger32	one of: -2, 1, 2,
poseX	8	IEEE754Double	<= 1.e+12, >= -1.e+12
poseY	8	IEEE754Double	<= 1.e+12, >= -1.e+12
poseYaw	8	IEEE754Double	in $[-\pi_d, \pi_d]$
distanceToLastLC	8	IEEE754Double	<= TYPE_MAX, >= 0.e+00
delay	8	IEEE754Double	<= TYPE_MAX, >= 0.e+00
progress	8	IEEE754Double	<= 1.e+00, >= 0.e+00
scan		Array	
–>length	4	UnsignedInteger32	<= 20000000, >= 0
–>elements	8	Position2DSingle	|x|,|y| <= 1.e+05
pathPoses		Array	
–>length	4	UnsignedInteger32	<= 20000000, >= 0
–>elements	12	Pose2DSingle	|x|,|y| <= 1.e+05
pathTypes		Array	
–>length	4	UnsignedInteger32	<= 20000000, >= 0
–>elements	4	SignedInteger32	one of: 0, 1, 2,

- **timestamp**: The time at which the Localization Client received the laser scan used to generate this Datagram.
- **visualizationId**: A unique id for matching datagrams that were emitted at the

same time but by different visualization interfaces.

- **status**: The recording status.
- **poseX**, **poseY**, **poseYaw**: Corresponding absolute coordinates in current map frame.
- **distanceToLastLC**: Distance traversed by the sensor since the last successful loop closure.
- **delay** : The relative delay within the system when processing laser scans. 0 : no delay; 1.0 : delay causes loop closures to be omitted when building the temporary map.
- **progress**: Currently unused.
- **scan**: A 2D point cloud representing the most recent laser scan, given in the current map frame.
- **pathPoints** : Array of 12 Byte Pose2DSingle, represents the path that the laser sensor took while capturing the recording.
- **pathTypes** : Array of 4 Byte integer of order state_1, state_2, using the ClientRecordingPathTypeEnum.

ClientRecordingRecordingStatus

Label	Value (SignedInteger32)	Description
STARTUP	-2	Visual Recording is starting up and not yet available
DELAYED	1	Visual Recording is delayed, the user should slow down the movement of the sensor
OK	2	Visual Recording ok

ClientRecordingPathTypeEnum

Label	Value (SignedInteger32)	Description
NORMAL	0	Path node of the current recording
POSSIBLE_LOOP_CLOSURE	1	Path node that should be revisited to make a loop closure

Label	Value (Signed-Integer32)	Description
POSSIBLY_VISIBLE_LOOP_-CLOSURE	2	POSSIBLE_LOOP_CLOSURE that lies within the field of view of the most recent laser scan

12.5 Mapping Output (Module: *ClientMap*)

12.5.1 Interfaces

ClientMapVisualization Interface The ClientMapVisualization interface provides information needed to visualize and evaluate the current map building process. It operates while the Localization Client is in mapping mode and uses the ClientMapVisualizationDatagram. This Datagram contains the most recently processed laser scan in the map reference frame. It also contains the estimated path taken by the sensor, and the path as estimated without loop closures. Additionally, the Datagram contains possible loop closure candidates, as well as additional information.

ClientMapMap Interface The ClientMapMap provides the user with the current state of the map that is being built. It operates while the Localization Client is in mapping mode and uses the MapDatagram.

12.5.2 Datagrams

ClientMapVisualizationDatagram

Name	Size (B)	Type	Restriction
timestamp	8	IEEE754Double	<= 1.e+12, >= 0.e+00
visualizationId	8	UnsignedInteger64	No restrictions.
status	4	SignedInteger32	one of: -2, 1, 2,
poseX	8	IEEE754Double	<= 1.e+12, >= -1.e+12
poseY	8	IEEE754Double	<= 1.e+12, >= -1.e+12
poseYaw	8	IEEE754Double	in $[-\pi_d, \pi_d]$
distanceToLastLC	8	IEEE754Double	<= TYPE_MAX, >= 0.e+00
delay	8	IEEE754Double	<= TYPE_MAX, >= 0.e+00

Name	Size (B)	Type	Restriction
progress	8	IEEE754Double	<= 1.e+00, >= 0.e+00
scan		Array	
–>length	4	UnsignedInteger32	<= 20000000, >= 0
–>elements	8	Position2DSingle	|x|,|y| <= 1.e+05
pathPoses		Array	
–>length	4	UnsignedInteger32	<= 20000000, >= 0
–>elements	12	Pose2DSingle	|x|,|y| <= 1.e+05
pathTypes		Array	
–>length	4	UnsignedInteger32	<= 20000000, >= 0
–>elements	4	SignedInteger32	one of: 0, 1, 2,

- **timestamp**: The time at which the Localization Client received the laser scan used to generate this Datagram.
- **visualizationId**: A unique id for matching datagrams that were emitted at the same time but by different visualization interfaces.
- **status**: The recording status.
- **poseX**, **poseY**, **poseYaw**: Corresponding absolute coordinates in current map frame.
- **distanceToLastLC**: Distance traversed by the sensor since the last successful loop closure
- **delay** : Currently unused.
- **progress**: Progress in building the map from the recording. The map will be complete when the progress reaches 1.0.
- **scan**: A 2D point cloud representing the most recent laser scan, given in the current map frame.
- **pathPoints** : Array of 12 Byte Pose2DSingle, represents the path that the laser sensor took while capturing the recording.
- **pathTypes** : Array of 4 Byte integer of order state_1, state_2, using the ClientRecordingPathTypeEnum.

ClientMapMappingStatus

Label	Value (Signed-Integer32)	Description
STARTUP	-2	Map building process is starting up and not yet running
DELAYED	1	Unsued
OK	2	Map building process is ok

ClientMapPathTypeEnum

Label	Value (Signed-Integer32)	Description
NORMAL	0	Path node of the used recording
POSSIBLE_LOOP_CLOSURE	1	Path node that would have enabled loop closures
POSSIBLY_VISIBLE_LOOP_-CLOSURE	2	POSSIBLE_LOOP_CLOSURE that was visible according to currently processed scan

12.6 Global Alignment (Module: *ClientGlobalAlign*)

12.6.1 Interfaces

ClientGlobalAlignVisualization Interface

The ClientGlobalAlignVisualization interface provides information needed to visualize and evaluate global alignment landmarks and their observations. It operates while the Localization Client is in the visual recording or mapping mode and uses the ClientGlobalAlignVisualizationDatagram. Note that this interface does not provide any data in the regular, non-visual recording mode.

12.6.2 Datagrams

ClientGlobalAlignVisualizationDatagram

Name	Size (B)	Type	Restriction
timestamp	8	IEEE754Double	<= 1.e+12, >= 0.e+00
visualizationId	8	UnsignedInteger64	No restrictions.
poses		Array	
–>length	4	UnsignedInteger32	<= 20000000, >= 0
–>elements	12	Pose2DSingle	|x|,|y| <= 1.e+05
landmarks		Array	
–>length	4	UnsignedInteger32	<= 20000000, >= 0
–>elements	var.	ClientGlobalAlign-LandmarkVisualization-Information	No restrictions.
observations		Array	
–>length	4	UnsignedInteger32	<= 20000000, >= 0
–>elements	8	ClientGlobalAlign-LandmarkObservation-Notice	No restrictions.

- **timestamp**: The time at which the Localization Client received the laser scan used to generate this Datagram.
- **visualizationId**: A unique id for matching datagrams that were emitted at the same time but by different visualization interfaces.
- **poses**: A set of poses previously visited by the platform. The platform may have observed landmarks from some of these poses.
- **landmarks**: A set of landmarks known to the system, specified through ClientGlobalAlign-LandmarkVisualizationInformation.
- **observations**: An array of ClientGlobalAlignLandmarkObservationNotice. Specifies which landmarks in the *landmarks* array were observed from which poses in the *poses* array. Note that some landmarks may have never been observed, and that some poses within the *poses* array may not be associated with any landmark.

ClientGlobalAlignLandmarkObservationNotice Encodes the fact that a landmark with the given index was observed from a pose with the given index.

Name	Size (B)	Type	Restriction
poseIndex	4	UnsignedInteger32	< 20000000, > 0

Name	Size (B)	Type	Restriction
landmarkIndex	4	UnsignedInteger32	< 20000000, > 0

ClientGlobalAlignLandmarkVisualizationInformation
Contains information about a landmark to be used for visualization.

Name	Size (B)	Type	Restriction
pose	12	Pose2DSingle	|x|,|y| <= 1.e+05
type	8	SignedInteger64	May only contain values defined within the ClientGlobalAlignObservationType.
hasOrientation	1	Boolean	No restrictions.
name		Array	
–>length	4	UnsignedInteger32	<= 241, >= 0
–>elements	1	PrintableAsciiCharacter	No restrictions.

- **pose**: Pose2DSingle encoding the pose of the landmark as estimated by the system.
- **type**: SignedInteger64 indicating the type of the landmark according to the ClientGlobalAlignObservationType.
- **hasOrientation**: Boolean indicating whether the orientation of the landmark is known. When set to false, the orientation angle of the landmark is undefined.
- **name**: Array of PrintableAsciiCharacter representing the unique name assigned to the landmark.

12.7 Sensor Input (Module: *ClientSensor*)

12.7.1 Interfaces

ClientSensorLaser Interface The ClientSensorLaser interface offers an alternative to the Localization Client's built-in laser scanner drivers. It allows the user to connect the client to an external source, which provides laser scan data. The Localization Client then uses these laser scans instead of the scans acquired through a built-in driver. To use this interface, set the client's `LaserComponent.laserType` to `simple` or

`simpletls` (TLS encryption). Note that the ClientSensorLaser interface acts as a consumer and will connect to an external TCP port opened by the user. The user must thus set the `LaserComponent.laserAddress` to the address and port of the external laser source. Once connected, this interface expects the user to send ClientSensorLaserDatagrams. Since the Laser Localization Software is running in docker, the `LaserComponent.laserAddress` has to be set to docker0 ip address instead of localhost ip address when the user program providing the laser data runs on the same machine.

Note that a scanner connected to the ClientSensorLaser interface–even though it can technically be any rotating laser scanner–still must provide performance similar to the officially supported laser scanners of the Laser Localization Software.

The ClientSensorLaser interface has to adapt some behaviors originated from the communication with the built-in laser scanner drivers. Once the TCP connection is established the first sent ClientSensorLaserDatagrams is only used as meta data for the laser configuration. The TCP connection is then closed. The real payload - the laser scans is sent after the second connection has been established.

In case of any network disturbances or system errors the recovery behavior of the laser software module is closing and reopening the TCP connection until the user TCP server is reachable and ready to provides laser data via the ClientSensorLaser interface.

The user has to consider these aspects while implementing the ClientSensorLaser interface. For implementation details, example codes in BinaryInterfaceServer.h are attached.

ClientSensorOdometry Interface The ClientSensorOdometry interface is a consumer that connects to an external odometry source. This source must provide pose information from a laser-independent, jump-free localization system. Such a source may rely on the vehicle odometry from the platform carrying the Localization Client. The Binary Interface will provide this data to the Localization Client. Once connected, this interface expects the user to send ClientSensorOdometryDatagrams. The user can enable this interface through the appropriate configuration entry. They can also set the address of the odometry source and toggle the use of TLS encryption this way. The user must also specify the transformation between the odometry reference frame and the position of the laser reference frame. If the odometry reference frame does not match the center of rotation of the mobile platform, this must be configured as well and the transformation between the center of rotation and the laser reference frame must be provided.

Odometry data supplied to this interface must adhere to the following error margins: The driving distance must be accurate to within 0.03 m plus 0.04 m per meter driven. Any jitter in the orientation must be well below 0.05 rad.

ClientSensorInertialMeasurementUnit Interface (beta version) The ClientSensorInertialMeasurementUnit interface is a consumer that connects to an external source. This source must provide pose information from a inertial measurement source. The Binary Interface will provide this data to the Localization Client. Once connected, this interface expects the user to send ClientSensorInertialMeasurementUnitDatagrams. The user can enable this interface through the appropriate configuration entry. They can also set the address of the IMU source and toggle the use of TLS encryption this way. The user must also specify the transformation between the inertial measurement frame and the reference frame of the laser sensor.

The error margins for data supplied by the inertial measurement unit are: The angular velocity must be correct up to 0.005 rad/s plus 0.04 rad/s per second.

Note: Although this interface available, the use of IMU data is currently a beta feature. Consequently, no guarantees are made when using this feature.

12.7.2 Datagrams

ClientSensorLaserDatagram This diagram describes a laser scan captured by a two-dimensional rotating laser scanner. The Localization Client assumes that scans are captured continuously and without gaps between them. This means that the duration of one rotation **duration_rotate** must equal to the time between two scans. In contrast, **duration_scan** is the time in which the scanner actually measures ranges during one rotation. Note that the scan number **scanNum** must increase by one for each transmitted scan.

The Laser Localization System uses right-handed coordinates. Thus, positive angles correspond to a counter-clockwise rotation, while negative angles represent clockwise rotations. This affects the **angleStart**, **angleEnd** and **angleInc** values within the ClientSensorLaserDatagram: For a scanner with a counter-clockwise scan direction, **angleStart** $<$ **angleEnd** and **angleInc** is positive. In contrast, for scanner with a clockwise scan direction **angleStart** $>$ **angleEnd** and **angleInc** is negative. Note that **duration_beam** will be greater than zero for either scan direction, as this is the time that passes between subsequent beams.

Name	Size (B)	Type	Restriction
scanNum	2	UnsignedInteger16	No restrictions.
time_start	8	IEEE754Double	<= 1.e+12, >= 0.e+00
uniqueId	8	UnsignedInteger64	No restrictions.
duration_beam	8	IEEE754Double	<= 1.e+12, >= 0.e+00
duration_scan	8	IEEE754Double	<= 1.e+12, >= 0.e+00
duration_rotate	8	IEEE754Double	<= 1.e+12, >= 0.e+00
numBeams	4	UnsignedInteger32	<= 100000, >= TYPE_MIN
angleStart	4	IEEE754Single	in $[-2\pi_f, 2\pi_f]$
angleEnd	4	IEEE754Single	in $[-2\pi_f, 2\pi_f]$
angleInc	4	IEEE754Single	in $[-2\pi_f, 2\pi_f]$
minRange	4	IEEE754Single	<= 1.e+04, >= 0.e+00
maxRange	4	IEEE754Single	<= 1.e+04, >= 0.e+00
ranges		Array	
–>length	4	UnsignedInteger32	<= 100000, >= 0
–>elements	4	IEEE754Single	<= 1.e+04, >= -1.e+04
hasIntensities	1	Boolean	boolean: 0 (false), 1 (true)
minIntensity	4	IEEE754Single	No restrictions.
maxIntensity	4	IEEE754Single	No restrictions.
intensities		Array	
–>length	4	UnsignedInteger32	<= 100000, >= 0
–>elements	4	IEEE754Single	No restrictions.

- **scanNum**: The scan number. Must increase by one between each subsequent scan.
- **time_start**: The time when the scanner begins its current rotation.
- **uniqueId**: A unique id that can be set to an arbitrary value. The ClientLocalizationPoseDatagram or ClientLocalizationVisualizationDatagram based on this scan will contain the specified value within their respective uniqueId fields. This way, scans can be exactly matched to their corresponding localization results.
- **duration_beam**: Time taken between two subsequent beams within a scan.
- **duration_scan**: Time taken for a single scan. Equal to the beam duration times the number of beams.
- **duration_rotate**: Time taken for a single, complete rotation of the scanner's scan head.
- **numBeams**: Number of beams within the scan.
- **angleStart**: Bearing of the first beam within the scan.
- **angleEnd**: Bearing of the last beam within the scan.
- **angleInc**: Angle between each subsequent beam. Must be equal to (**angleEnd** -

angleStart) / (**numBeams** - 1). Positive values indicate a scanner with a counter-clockwise scan direction. Use negative values for a scanner with a clockwise rotation.

- **minRange**: Lowest possible range which can be measured by the scanner. Measurements below this value will be treated as erroneous and discarded.
- **maxRange**: Highest possible range which can be measured by the scanner. Measurements above this value will be treated as erroneous and discarded.
- **ranges**: Array of ranges as measured by the scanner, each element corresponding to one beam. Thus, the first element within this array corresponds to the first beam measured by the scanner within a given scan. The length of this array must be equal to **numBeams**. A value of -1 indicates that the corresponding beam was not reflected by any object within the scanners range. Other negative values indicate measurement errors. Possible error sources include blinding lights (indicated by a measurement value of -2), a dirty scanner window (indicated by a value of -4), or an internal scanner error (indicated by a value of -3).
  Note that there is a difference between a beam with no reflection and a measurement error: In the former case, the scanner has detected free space within the environment. In the latter case, the scanner provides no information about the environment. Thus the measurement for a beam that was not reflected by a target should always be set to -1. It is not sufficient to merely report a very large measurement that lies beyond the value of the configuration parameter **config_laser.mask.max_range** or beyond the **maxRange**. Such a measurement would be discarded as erroneous, which is different from a measurement that indicates free space.
- **hasIntensities**: If this is false, the length of the **intensities** array can be zero. Else, it must be the same size as the **ranges** array.
- **minIntensity**: Lowest possible intensity value which can be measured by the scanner. This field is mandatory even if **hasIntensities** is false.
- **maxIntensity**: Highest possible intensity value which can be measured by the scanner. This field is mandatory even if **hasIntensities** is false.
- **intensities**: Array of beam reflection intensities, as measured by the scanner. Each element corresponding to one beam. This field is mandatory even if **hasIntensities** is false.

ClientSensorOdometryDatagram This Datagram describes a pose estimate derived via vehicles odometry or a similar jump-free, external localization source. To specify the transformation from the odometry frame to the laser frame, the user must set the corresponding configuration entry.

Name	Size (B)	Type	Restriction
timestamp	8	IEEE754Double	<= 1.e+12, >= 0.e+00
odoNumber	4	UnsignedInteger32	No restrictions.
epoch	8	UnsignedInteger64	No restrictions.
x	8	IEEE754Double	<= 1.e+12, >= -1.e+12
y	8	IEEE754Double	<= 1.e+12, >= -1.e+12
yaw	8	IEEE754Double	in $[-\pi_d, \pi_d]$
v_x	8	IEEE754Double	<= TYPE_MAX, >= TYPE_MIN
v_y	8	IEEE754Double	<= TYPE_MAX, >= TYPE_MIN
omega	8	IEEE754Double	<= TYPE_MAX, >= TYPE_MIN
velocitySet	1	Boolean	boolean: 0 (false), 1 (true)

- **timestamp**: The time when this pose estimate of the platform is generated. Note that each Datagram should be transmitted as soon as possible. Consequently, **timestamp** should be close to the current time, except for a small difference allowed for computing and transmitting the Datagram.
- **odoNumber**: Must be incremented between each subsequent Datagram being sent.
- **epoch**: Subsequent datagrams within the same epoch will be treated as locally precise and coherent. If there is a gap within the odometry data being sent, the epoch must be incremented. It should also be incremented if the odometry data does not match the actual motion of the platform.
- **x**, **y**, **yaw**: The Cartesian (x,y) coordinates and yaw angle that specify the platform's pose at the time given by **timestamp**. The origin and orientation of the underlying reference frame is irrelevant, as long as all Datagrams with the same **epoch** value use the same frame.
- **v_x**, **v_y**, **omega**: Linear velocity along the x and y axis and angular velocity in yaw. As measured at the time **timestamp** and given in the platform reference frame.
- **velocitySet**: Indicates whether or not the velocity is known. This must be set to false if the velocity is unknown. In this case, the velocity entries within the Datagram will be ignored.

ClientSensorInertialMeasurementUnitDatagram This Datagram contains data measured by a vehicle's on-board inertial measurement unit. To specify the transformation from the measurement unit frame to the laser frame, the user must set the corresponding configuration entry. Note that all measurements may be given in an arbitrary reference frame, as long as consecutive messages use the same frame.

Name	Size (B)	Type	Restriction
timestamp	8	IEEE754Double	<= 1.e+12, >= 0.e+00
imuNumber	4	UnsignedInteger32	No restrictions.
epoch	8	UnsignedInteger64	No restrictions.
orientation_qw	8	IEEE754Double	<= 1.e+00, >= -1.e+00
orientation_qx	8	IEEE754Double	<= 1.e+00, >= -1.e+00
orientation_qy	8	IEEE754Double	<= 1.e+00, >= -1.e+00
orientation_qz	8	IEEE754Double	<= 1.e+00, >= -1.e+00
omega_x	8	IEEE754Double	<= TYPE_MAX, >= TYPE_MIN
omega_y	8	IEEE754Double	<= TYPE_MAX, >= TYPE_MIN
omega_z	8	IEEE754Double	<= TYPE_MAX, >= TYPE_MIN
a_x	8	IEEE754Double	<= TYPE_MAX, >= TYPE_MIN
a_y	8	IEEE754Double	<= TYPE_MAX, >= TYPE_MIN
a_z	8	IEEE754Double	<= TYPE_MAX, >= TYPE_MIN
orientationSet	1	Boolean	boolean: 0 (false), 1 (true)
linearAccSet	1	Boolean	boolean: 0 (false), 1 (true)

- **timestamp**: The time when the measurements within this Datagram are taken. Note that each Datagram should be transmitted as soon as possible. Consequently, **timestamp** should be close to the current time, except for a small difference allowed for computing and transmitting the Datagram.
- **imuNumber**: Must be incremented between each subsequent Datagram being sent.
- **epoch**: Subsequent datagrams within the same epoch will be treated as locally precise and coherent. If there is a gap within the IMU data being sent, the epoch must be incremented. It should also be incremented if the measurements do not match the actual motion of the platform.
- **orientation_qw**, **orientation_qx**, **orientation_qy**, **orientation_qz**: The orientation of the IMU at the time **timestamp**, in quaternion representation.
- **omega_x**, **omega_y**, **omega_z**: The angular velocities measured by the IMU around the three axes at the time **timestamp**.
- **a_x**, **a_y**, **a_z**: The linear acceleraration measured by the IMU around the three axes at the time **timestamp**.
- **orientationSet**: Indicates whether or not the IMU's orientation is known. Must be set to false if the orientation is unknown. In this case, the corresponding entries within the Datagram will be ignored.
- **linearAccSet**: Indicates whether or not the IMU's linear accelerations are known. Must be set to false if the accelerations are unknown. In this case, the corresponding entries within the Datagram will be ignored.

12.8 FAQ

How to encode / decode base64 strings or data ? Following snippet uses the cppcodec library (https://github.com/tplgy/cppcodec, released under MIT license, Copyright (c) 2015 Topology Inc., Copyright (c) 2018 Jakob Petsovits, Copyright (c) various other contributors, see individual files) to encode, decode strings

```
std::string originalString{"HelloWorld!"};
std::string encodedString
    {cppcodec::base64_rfc4648::encode(originalString)};
std::string decodedString
    {cppcodec::base64_rfc4648::decode(encodedString)};

std::cout << "originalString: " << originalString << std::endl;
std::cout << "encodedString: " << encodedString << std::endl;
std::cout << "decodedString: " << decodedString << std::endl;
```

Which ports are used for the communication? The interfaces provided by the Laser Localization Software use the following TCP ports:

Interface	Port	Secure Port (TLS)
JSON RPC Localization Client	8080	8443
JSON RPC Map Server	8082	8445
JSON RPC Localization Client Support	8084	8447
JSON RPC Map Server Support	8086	8449
Binary ClientControlMode	9004	9444
Binary ClientMapMap	9005	9445
Binary ClientMapVisualization	9006	9446
Binary ClientRecordingMap	9007	9447
Binary ClientRecordingVisualization	9008	9448
Binary ClientLocalizationMap	9009	9449
Binary ClientLocalizationVisualization	9010	9450
Binary ClientLocalizationPose	9011	9451
Binary ClientGlobalAlignVisualization	9012	9452
Internal: Map Server Ports for Localization Client Communication		21638-21643

How to send JSON-RPC messages? JSON RPC messages are transmitted using a transport independent remote procedure call protocol. Consequently, JSON RPC Messages can be sent to the components using several different programming languages or tools. For example, the following command sends a sessionLogin message to a product element using the command-line tool curl:

```
curl --header "Content-Type: application/json" --request POST --data
'{"jsonrpc":"2.0","method":"sessionLogin","params":{"query":{
  "timeout": {"valid" : true, "time" : 600, "resolution" : 1},
  "userName": "John Armstrong", "password": "mySecurePassword"}
  },"id":1}' http://myClienttHost:myClientPort
```

What do the letters 'T' and 'Z' in timestamp mean? The timestamp e.g. 20200330T184520Z has the format according to ISO 8601 basic format in UTC. The letter 'T' is the time designator and is located before the time element. The letter 'Z' refers to the Zulu time zone.

13 User Groups

This product includes a user management with the following predefined user groups:

- observer: limited read access, cannot modify system
- user: is able to do customer-specific configuration to make use of the system
- admin: same as *user*, with additional rights to manage customer-specific users

The following table shows the group-dependent access of the API-calls in alphabetic order.

API	admin	user	observer	comment
aboutBuildList	x	x	x	all
aboutModulesList	x	x	x	all including not authenticated user
certificateSet	x			admin
clientGlobalAlignAddObservation	x	x		user, admin
clientGlobalAlignDeleteObservations	x	x		user, admin
clientGlobalAlignListObservations	x	x	x	all
clientGlobalAlignReplaceObservations	x	x		user, admin
clientLaserMaskingGetScan	x	x		user, admin

API	admin	user	observer	comment
clientLaserMaskingStart	x	x		user, admin
clientLaserMaskingStop	x	x		user, admin
clientLocalizationSetSeed	x	x		user, admin
clientLocalizationStart	x	x		user, admin
clientLocalizationStop	x	x		user, admin
clientManualAlignGetPointCloud	x	x		user, admin
clientManualAlignSet	x	x		user, admin
clientManualAlignStart	x	x		user, admin
clientManualAlignStop	x	x		user, admin
clientMapDelete	x	x		user, admin
clientMapList	x	x	x	all
clientMapRename	x	x		user, admin
clientMapSend	x	x		user, admin
clientMapStart	x	x		user, admin
clientMapStop	x	x		user, admin
clientRecordingDelete	x	x		user, admin
clientRecordingStart	x	x		user, admin
clientRecordingStartVisualRecording	x	x		user, admin
clientRecordingStop	x	x		user, admin
clientRecordingStopVisualRecording	x	x		user, admin
clientRecordingList	x	x	x	all
clientRecordingRename	x	x		user, admin
clientRecordingSetCurrentPose	x	x		user, admin
clientUserAdd	x			admin
clientUserChangePassword	x			admin
clientUserChangeOwnPassword	x	x	x	all
clientUserDelete	x			admin
clientUserList	x			admin
clientUserListGroup	x			admin
configList	x	x	x	all
configSet	x	x		user, admin
diagnosticClear	x	x		user, admin
diagnosticList	x	x	x	all
licensingFeatureGet TrustedPlatformModuleInformation	x	x		user, admin
licensingFeatureGetHostId	x	x		user, admin
licensingFeatureList	x	x	x	all
licensingFeatureSet	x	x		user, admin

API	admin	user	observer	comment
serverMapDelete	x	x		user, admin
serverMapGetImage	x	x	x	all
serverMapGetImageWithResolution	x	x	x	all
serverMapGetInfo	x	x	x	all
serverMapGetMapThumbnail	x	x	x	all
serverMapGetPointCloud	x	x	x	all
serverMapList	x	x	x	all
serverMapRename	x	x		user, admin
serverUserAdd	x			admin
serverUserChangePassword	x			admin
serverUserChangeOwnPassword	x	x	x	all
serverUserDelete	x			admin
serverUserList	x			admin
serverUserListGroup	x			admin
serverInternalFleetConfigSet	x	x	x	all
serverInternalFleetConfigGet	x	x	x	all
sessionGroupInfo	x	x	x	all
sessionLogin	x	x	x	all including not authenticated user
sessionLogout	x	x	x	all
sessionRefresh	x	x	x	all
supportReportCreate	x	x		user, admin
supportReportCreateMinimal	x	x		user, admin
supportReportDelete	x	x		user, admin
supportReportGetPath	x	x		user, admin
supportReportList	x	x	x	all
supportReportSetDescription	x	x		user, admin
supportRecoveryCreate	x			admin
supportRecoveryDelete	x			admin
supportRecoveryFrom	x			admin
supportRecoveryFactoryReset	x			admin
supportRecoveryList	x			admin
systemShutdown	x			admin

14 Licensing

The API and system functionality that is available to the user depends on the software features for which are licensed.

14.1 Methods for host machine identification

Licensing Method	Description
WIBUDONGLE	Use a Wibu dongle attached to the host machine as identifier.
TPM	Use the host machine's trusted platform module for identification.

14.2 Licensing for API methods

When calling the API methods described in this document, the following licenses are required:

Method name	Required feature license
aboutBuildList	None (always allowed)
aboutModulesList	None (always allowed)
certificateSet	None (always allowed)
clientGlobalAlignAddObservation	None (always allowed)
clientGlobalAlignDeleteObservations	None (always allowed)
clientGlobalAlignListObservations	None (always allowed)
clientGlobalAlignReplaceObservations	None (always allowed)
clientLaserMaskingGetScan	None (always allowed)
clientLaserMaskingStart	None (always allowed)
clientLaserMaskingStop	None (always allowed)
clientLocalizationSetSeed	None (always allowed)
clientLocalizationStart	Feature "Localization"
clientLocalizationStop	None (always allowed)
clientManualAlignGetPointCloud	None (always allowed)
clientManualAlignSet	None (always allowed)
clientManualAlignStart	None (always allowed)
clientManualAlignStop	None (always allowed)
clientMapDelete	None (always allowed)
clientMapList	None (always allowed)
clientMapRename	None (always allowed)

Method name	Required feature license
clientMapSend	Feature ”Map Creation”
clientMapStart	Feature ”Map Creation”
clientMapStop	None (always allowed)
clientRecordingDelete	None (always allowed)
clientRecordingList	None (always allowed)
clientRecordingRename	None (always allowed)
clientRecordingSetCurrentPose	None (always allowed)
clientRecordingStart	None (always allowed)
clientRecordingStartVisualRecording	Feature ”Visual Recording”
clientRecordingStop	None (always allowed)
clientRecordingStopVisualRecording	None (always allowed)
clientUserAdd	None (always allowed)
clientUserChangePassword	None (always allowed)
clientUserChangeOwnPassword	None (always allowed)
clientUserDelete	None (always allowed)
clientUserList	None (always allowed)
clientUserListGroup	None (always allowed)
configList	None (always allowed)
configSet	None (always allowed)
diagnosticClear	None (always allowed)
diagnosticList	None (always allowed)
licensingFeatureGetTrustedPlatformModuleInformation	None (always allowed)
licensingFeatureGetHostId	None (always allowed)
licensingFeatureList	None (always allowed)
licensingFeatureSet	None (always allowed)
serverMapDelete	None (always allowed)
serverMapGetImage	None (always allowed)
serverMapGetImageWithResolution	None (always allowed)
serverMapGetInfo	None (always allowed)
serverMapGetMapThumbnail	None (always allowed)
serverMapGetPointCloud	None (always allowed)
serverMapList	None (always allowed)
serverMapRename	None (always allowed)
serverInternalFleetConfigSet	None (always allowed)
serverInternalFleetConfigGet	None (always allowed)
serverUserAdd	None (always allowed)
serverUserChangePassword	None (always allowed)
serverUserChangeOwnPassword	None (always allowed)

Method name	Required feature license
serverUserDelete	None (always allowed)
serverUserList	None (always allowed)
serverUserListGroup	None (always allowed)
sessionGroupInfo	None (always allowed)
sessionLogin	None (always allowed)
sessionLogout	None (always allowed)
sessionRefresh	None (always allowed)
supportRecoveryCreate	None (always allowed)
supportRecoveryDelete	None (always allowed)
supportRecoveryFrom	None (always allowed)
supportRecoveryFactoryReset	None (always allowed)
supportRecoveryList	None (always allowed)
supportReportCreate	None (always allowed)
supportReportCreateMinimal	None (always allowed)
supportReportDelete	None (always allowed)
supportReportGetPath	None (always allowed)
supportReportList	None (always allowed)
supportReportSetDescription	None (always allowed)
systemShutdown	None (always allowed)

14.3 Licensing and the Localization Client

Operating the Localization Client in some Client Control Modes may also require a license:

ClientMode	Required feature license
VISUALRECORDING	Feature ”Visual Recording”
MAP	Feature ”Map Creation”
LOC	Feature ”Localization”
REC	None (always allowed)
ALIGN	None (always allowed)
MASK	None (always allowed)

14.3.1 Licensing and specific Client features

If the ”Global Alignment” feature is not licensed, the API methods from the ClientGlobalAlign module will function nevertheless. Landmarks used by global alignment can thus still be created and manipulated even without a license for the ”Global Alignment” feature. However, these landmarks are ignored by the client when building the map, unless the ”Global Alignment” feature is licensed. In this case, the client will publish a warning using the Diagnostic system.

14.4 Licensing and the Map Server

Operating the Map Server does not require a license.

14.5 Licensing and the Support Component

Operating the Support Component does not require a license.

15 Appendix

15.1 Code Examples

15.1.1 Binary Interface

The following examples are shipped together with this document. They demonstrate how to communicate with some of the Binary Interfaces using C++11. These examples use Linux sockets and the Linux implementation of the TCP/IP protocol. They have been tested using the GNU Compiler Collection.

Filename	Function
BinaryInterfaceServer.h	Implementation of a Binary Interface Server
BinaryInterfaceClient.h	Implementation of a Binary Interface Client
BinaryInterfacePoseStruct.h	Definition of the ClientLocalizationPoseDatagram
BinaryInterfaceDummyLaserData.h	Implementation of a dummy-filled ClientSensorLaserDatagram

Filename	Function
TestClientLocalizationPose.cpp	Program testing the ClientLocalizationPose interface
TestClientSensorLaser.cpp	Program testing the ClientSensorLaser interface

16 Changelog

This section gives a brief overview on the changes made for each release. Due to the scope of the API, it is not guaranteed to be complete. The module version numbers indicate whether or not a specific API module has undergone changes. If this is the case, users should check the methods, messages, and interfaces associated with that module carefully.

16.1 Version 1.2

The 1.2 release again contains several improvements to the API of the Laser Localization Software. As each of these changes only affects an indivdual module, this list is grouped accordingly. Based on customer feedback, this document itself has also been extended with more detailed information. Besides changes to the API itself, this changelog therefore also points out some of the improvements made to this document.

16.1.1 Clarifications regarding user interfaces

The sections about the JSON RPC and binary interfaces now specify the behavior of these interfaces in case of input validation failure.

16.1.2 Changes to the ClientGlobalAlign module

- clientGlobalAlignReplaceObservations will no longer generate excessively long recording names. Instead, the names of the modified recordings are truncated to meet the requirements of a RecordingName.
- clientGlobalAlignReplaceObservations will now return the new `CLIENTGLOBALALIGN_OBSERVATION_NOT_FOUND` error when trying to replace a non-existing observation.

- clientGlobalAlignDeleteObservations will now return the new `CLIENTGLOBALALIGN_OBSERVATION_NOT_FOUND` error when trying to replace a non-existing observation.
- Pointed out the possibility of changing an observation's type to `IGNORE` instead of deleting it with clientGlobalAlignDeleteObservations.

16.1.3 Changes to the ClientLocalization module

- The `age` field within the ClientLocalizationPoseDatagram may now contain negative values. Such values may by caused by improperly-synchronized clocks, as explained in the datagram's description.
- The ClientLocalizationPoseDatagram now contains information about the uncertainty of the current pose estimate. This information is given as an approximated covariance matrix and is encoded by the fields `covariance_1,1` to `covariance_3,3`.
- The `uniqueId` field within the ClientLocalizationPoseDatagram now functions properly, and is no longer marked as unused.
- Added additional information regarding the effects of calling the clientLocalizationSetSeed method while the Localization Client is not receiving laser sensor data.

16.1.4 Changes to the ClientManualAlign module

- clientManualAlignSet will now return the name of the aligned map using the new ClientManualAlignMapNameResponseMessage.
- clientManualAlignSet will no longer generate excessively long map names. Instead, map names are truncated to meet the requirements of a ClientMapName.

16.1.5 Changes to the ClientSensor module

- The fields `minRange` and `maxRange` of the ClientSensorLaserDatagram are now interpreted correctly. Previously, the contents of these fields were erroneously ignored by the LLS.
- Within the `ranges` array of the ClientSensorLaserDatagram, it is now possible to specify why a particular beam emitted by the laser scanner failed to measure a valid range. This allows the Localization Client to better handle such measurement errors. The corresponding section contains details on the various types of measurement errors that may be reported by laser scanners and how to encode them wthin the `ranges` array.

- The `uniqueId` field within the ClientSensorLaserDatagram now functions properly, and is no longer marked as unused.
- Added general information on how to communicate with the ClientSensorLaser interface.
- The section on the ClientSensorLaserDatagram now contains additional information about the reference frame used by the Laser Localization Software. This includes enhanced information on how to encode data from laser scanners with both clockwise and counter-clockwise scanning directions.

16.1.6 Changes to the ClientUser module

While no changes were made to this module, the module version number was incremented. This was necessary due to changes made in version 1.1, which had previously been overlooked. Refer to the changelog for version 1.1 for details.

16.1.7 Changes to the Config module

- Fixed a bug that could cause the Localization Client to hang after calling the configSet API method.
- The following parameters in ConfigEntries have been added:
  - `locator_graph.odom_error_model.laser_T_odo.{x,y,z,roll,pitch,yaw}`
  - `locator_graph.odom_error_model.laser_T_odo.enabled`
  - `locator_graph.imu_error_model.laser_T_imu.{x,y,z,roll,pitch,yaw}`
  - `locator_graph.imu_error_model.laser_T_imu.enabled`
- The following parameters in ConfigEntries have been removed:
  - `imu_odo_error_model.lever_arm_m`
  - `odo_error_model.lever_arm_m`
  - `global_strategies.lidar_odometry_filter_mode`

16.1.8 Changes to the LicensingFeature module

- Removed the field `endorsementPublicKey` from the LicensingHostTpmResponseMessage sent by the licensingFeatureGetTrustedPlatformModuleInformation method.
- Added the field `hardwareInformation` from the LicensingHostTpmResponseMessage sent by the licensingFeatureGetTrustedPlatformModuleInformation method.
- Provided additional information on the meaning of the LicensingStatus, LicensingType and LicensingFeatureCapability objects.

16.1.9 Changes to the Diagnostic module

- Added additional information on assembling the text contained within each `DiagnosticEntry`.
- The response code module identifier for the Diagnostic module was changed from the already-assigned value of 0x0002 to 0x0008. This currently has no effect as there are now response codes specific to said module.

16.1.10 Changes to the ServerMap module

- Provided additional information about the `MAPimageOrigin` field within the ServerMapPngImageResponseMessage

16.1.11 New module: ServerInternal

This module has been newly introduced in version 1.2. As an internal module, it should not be used by the end user. However, the module has been fully documented here for the sake of completeness.

16.1.12 Changes to the ServerUser module

While no changes were made to this module, the module version number was incremented. This was necessary due to changes made in version 1.1, which had previously been overlooked. Refer to the changelog for version 1.1 for details.

16.2 Version 1.1

The Laser Localization Software API received numerous additions and improvements for the 1.1 release. This section first lists general changes that affect multiple modules and then discusses module-specific changes.

16.2.1 Network ports for Client-Server communication

The network traffic between the Localization Client and Map Server is now encrypted. As a result, the TCP ports used have changed. The new ports are given in the table of ports.

16.2.2 Clarifications regarding licensing

The Licensing section now gives detailed information about which features have to be licensed to use the API methods and functionalities described in this document.

16.2.3 Response Codes

Many API methods now make better use of the existing response codes. These methods will now return specific codes such as `FILE_ACCESS_FAILED` where they would previously only return an `INTERNAL_ERROR`. The methods that implement this improved behavior include:

- clientGlobalAlignReplaceObservations
- clientGlobalAlignDeleteObservations
- clientMapStart
- clientMapSend

16.2.4 Protecting entities from inadvertent deletion

Some API methods that create entities such as recordings or maps would previously overwrite any pre-existing entity with the same name. To avoid the accidental loss of data, these cases should now result in an `ENTITY_ALREADY_EXISTS` error instead. If the user wishes to replace an existing recording, map, or similar entity, they should first delete it using the appropriate API method. Methods which should no longer overwrite existing entities include:

- clientMapStart: Will no longer overwrite existing maps.
- clientRecordingRename: Will no longer overwrite existing recordings.
- clientMapRename: Will no longer overwrite existing client-side maps.

16.2.5 Protecting entities in use

API methods that delete or rename entities such as recordings or maps will no longer do so if the target entity is in use. Instead, these methods now return the newly-introduced `ENTITY_IN_USE` error. To delete the affected entity, first stop the procedure that is currently using it. Affected methods include:

- clientRecordingRename: Will no longer rename recordings that are currently being created. Will no longer rename recordings from which a map is currently being built.
- clientRecordingDelete: Will no longer delete recordings that are currently being created. Will no longer delete recordings from which a map is currently being built.

16.2.6 Ensuring coordinate precision

To ensure that floating-point coordinates received and sent by the system are sufficiently precise, the range of such coordinates has been restricted: Cartesian coordinates that are sent to the system may no longer exceed values of 10,000 meters in the X and Y axis. This limit also applies to the map transformations used by clientManualAlignSet. Laser range measurements sent through the ClientSensorLaser interface may also not exceed 10,000 meters. In return, the system will never generate cartesian coordinates with absolute values exceeding 100,000 meters.

16.2.7 Handling irrational numbers

The irrational number π cannot be accurately represented through IEEE-754 floating point numbers. The range limits specified by the binary interfaces now account for this fact, as described in a section on common constants.

16.2.8 Visualization Ids

A visualizationId now allows the user to synchronize visualization information from different interfaces. In particular, this allows users to synchronize ClientRecordingVisualizationDatagram and ClientMapVisualizationDatagram with the new ClientGlobalAlignVisualizationDatagram.

In return, the previously-unused uniqueId was removed from the ClientRecordingVisualizationDatagram and ClientMapVisualizationDatagram.

16.2.9 Changes to the LicensingFeature module

- This module has been extended significantly, adding support for licensing through WIBU dongles and Trusted Platform Module 2.0.

  - Specific additions include the licensingFeatureGetTrustedPlatformModuleInformation method to retrieve information about the Trusted Platform Module, as well as several new, module-specific response codes.
- In return, the old HWSERIAL hardware serial licensing method has been retired.

16.2.10 Changes to the Config module

- The configSet API method now only gives a response after the component's internal configuration has been updated. During this time, additional API method calls may be rejected with a `NOT_IN_REQUIRED_STATE` response.

16.2.11 Changes to the Certificate module

- New module-specific response codes have been added for better error handling.

16.2.12 Changes to the ClientRecording module

- The ClientRecordingVisualizationDatagram now contains a visualization Id, which allows synchronization with the new ClientGlobalAlignVisualizationDatagram.
- The user can use the clientRecordingSetCurrentPose method to set the current pose of the platform during a recording. This is especially useful if the platform starts the recording run with a known pose.
- The strings "." or ".." are no longer valid RecordingNames.

16.2.13 Changes to the ClientMap module

- A mapping process started by clientMapStart can now be aborted by calling the new clientMapStop method.
- The clientMapSend method will now report communication issues using the new `CLIENTMAP_SERVER_UNREACHABLE` and `CLIENTMAP_SERVER_COMMUNICATION_FAILED` response codes.
- If the map could not be sent because the client failed not read it, the method will now return a `FILE_ACCESS_FAILED` error instead.
- The ClientMapVisualizationDatagram now contains a visualization Id, which allows synchronization with the new ClientGlobalAlignVisualizationDatagram.
- The strings "." or ".." are no longer valid ClientMapNames.

16.2.14 Changes to the ClientManualAlign module

- The value range of map transformations that can be sent to clientManualAlignSet has been restricted. Refer to the method documentation for details.

16.2.15 Changes to the ClientGlobalAlign module

- This module and its associated functionality are new additions to the API.

16.2.16 Changes to the ClientUser module

- Users can now change their own passwords using the new clientUserChangeOwnPassword method, which requires the user's current password.
- Users not in the admin group may no longer call clientUserChangePassword, and must use clientUserChangeOwnPassword instead.

16.2.17 Changes to the ServerMap module

- The strings "." or ".." are no longer valid ServerMapNames.

16.2.18 Changes to the ServerUser module

- Users can now change their own passwords using the new serverUserChangeOwnPassword method, which requires the user's current password.
- Users not in the admin group may no longer call serverUserChangePassword, and must use serverUserChangeOwnPassword instead.

16.2.19 Changes to the SupportReport module

- This module now contains the supportReportCreateMinimal method, which generates a minimal support report. Such a report contains limited information, but also has a smaller file size.
- Support reports are now compressed and thus stored under a `.tar.bz2` instead of `.tar` file extension.

16.2.20 Changes to the SupportRecovery module

- New module-specific response codes have been added for better error handling.
- The new method supportRecoveryFactoryReset allows the user to perform a factory reset.

16.3 Version 1.0

The Bosch Rexroth Laser Localization Software interface described in this document has undergone major changes in nearly all areas. It is thus impractical to give a complete and detailed log of all changes made for the release of version 1.0. Instead, users should study the new documentation carefully, even if they are already familiar with older, development versions of this API.