rexroth
A Bosch Company

API Documentation

Rexroth ROKIT Locator

Version 1.10.5

Bosch Rexroth AG, Version 31.10.2024, EN, RB13109422

Contents

1 Preface

This is the interface documentation for the Rexroth ROKIT Locator Version 1.10.5.

1.1 GENERAL NOTES

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, 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 interfaces used by the ROKIT Locator. It supplements the ROKIT Locator manual, which describes the overall system, its capabilities, and how to use it. In contrast, this interface documentation specifies how exactly the ROKIT Locator's interfaces should be used. Consequently, the user should first study the software manual to understand the ROKIT Locator. They should then refer to this document when actually writing software that interacts with the ROKIT Locator. 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 NAMING CONVENTIONS

- 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 or frames, 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 frame, gets the name of the frame 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.
- A ROKIT Locator system consists of a ROKIT Locator Server, at least one ROKIT Locator Client, and one or more optional ROKIT Locator Support. 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.

2 API Modules

The interface of the ROKIT Locator 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, session etc.). Each module may also encompass one or more Binary Interfaces and their associated data types. Additionally, a module may also include user-configurable parameters which govern the underlying functionality of that module. Unless explicitly stated otherwise, changing the value of a parameter will only affect the behavior of methods and interfaces from the same module.

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. API 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 ROKIT Locator should always check the actual API module versions. This ensures that the program is compatible with this specific instance of the ROKIT Locator.

The name of an API module is a string which will remain unchanged throughout all future ROKIT Locator releases. If the task of the API module is available for both the ROKIT Locator Client and the ROKIT Locator 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 component ROKIT Locator Support bear the prefix *Support*. Note that ROKIT Locator Support only contains a reduced subset of the common modules. Besides the SupportReport and SupportRecovery modules exclusive to ROKIT Locator Support, it only implements the modules AboutModules, AboutBuild, DataExchange, System, 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, Datagram or parameter 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.

Note that the *Config* module is used to set the parameters of all other modules. As such, a change to the parameters of any other module will also affect the version of the Config module itself. Here, removing a parameter, changing its type, the values it accepts, or its meaning leads to an increase in the *MajorNumber*. However, adding a new parameter only increases the *MinorNumber*, as these new parameters can be ignored. Similarly, a change in the default value of a parameter also increments the *MinorNumber*.

Refer to the changelog for information on how these modules changed with each software release.

2.2 API MODULE OVERVIEW

2.2.1 Common API Modules

These API modules are common in multiple Rexroth ROKIT products and will behave the same (for the same module version).

Name of API module	Defines	Version
	Description	
AboutModules	RPC	5.0
	get names and versions of all API-Modules which are supported from the current ROKIT Locator version	
Session	RPC	4.0
	manage access to the ROKIT Locator	
Diagnostic	RPC, Binary	7.0
	get diagnostic information	
Licensing	RPC, Config	7.2
	handle licenses and get information about them	
Config	RPC, *Config*	8.0
	configure the ROKIT Locator	
AboutBuild	RPC	3.1
	get information about the ROKIT Locator	
Certificate	RPC, Config	3.0
	handle the certificate management	
System	RPC, Binary	3.2
	general control over the ROKIT Locator state	
User	RPC	1.0
	manage users within the ROKIT Locator	
Internal	RPC	1.1
	storage of user specific configurations (internal use)	
SupportReport	RPC, Config	4.1
	provides ROKIT Locator facilities used for customer support	
SupportRecovery	RPC, Config	4.1
	manage ROKIT Locator recovery, backup and update	

(table continued)

Name of API module	Defines	Version
	Description	
DataExchange	RPC, Binary	1.0
	exchange data between ROKIT Locator instances	

2.2.2 ROKIT Locator Client API Modules

These API modules are only available in the Rexroth ROKIT Locator Client.

Name of API module	Defines	Version
	Description	
ClientApplication	Config	1.2
	Functionality affecting the overall behavior of the ROKIT Locator Client	
ClientControl	Binary	3.2
	handles the non mode-specific control of the ROKIT Locator Client including current mode output	
ClientRecording	RPC, Binary	5.0
	start/stop recording and visual recording, manage recordings	
ClientMap	RPC, Binary	4.1
	start and visualize offline mapping, manage maps and get information about them	
ClientLocalization	RPC, Binary, Config	7.2
	start/stop and visualize localization	
ClientManualAlign	RPC	5.0
	manually align a constructed map	
ClientGlobalAlign	RPC, Binary	4.0
	enables global map consistency using landmarks	
ClientLaserMask	Config	6.1
	mask out static objects in the field of view of the laser	
ClientSensor	RPC, Binary, Config	7.1
	handles the input and output of sensor data provided by the customer	

(table continued)

Name of API module	Defines	Version
	Description	
ClientUser **(deprecated)**	RPC	4.0
	manage users on a ROKIT Locator Client (this API module is deprecated, use the common API module User instead)	
ClientExpandMap	RPC, Binary	4.0
	expand existing map	

2.2.3 ROKIT Locator Server API Modules

These API modules are only available in the Rexroth ROKIT Locator Server.

Name of API module	Defines	Version
	Description	
ServerMap	RPC, Config	8.0
	manage the central maps and get information about them	
ServerPostAlign	RPC, Config	1.0
	manually align scans within a map after mapping	
ServerUser **(deprecated)**	RPC	4.0
	manage users on a ROKIT Locator Server (this API module is deprecated, use the common API module User instead)	
ServerInternal **(deprecated)**	RPC	3.1
	storage of internal network configurations for the management of the ROKIT Locator Fleet (internal use)(this API module is deprecated, use the common API module Internal instead)	

2.2.4 ROKIT Locator Server Support API Modules

These API modules are only available in the Rexroth ROKIT Locator Server Support.

Name of API module	Defines	Version
	Description	
SupportFleetinfo	RPC	1.0
	provides ROKIT Locator facilities used for fleet info	

3 Interfaces

The ROKIT Locator provides the following 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 ROKIT Locator Client or ROKIT Locator 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 ROKIT Locator for a list of all interfaces provided by the system.

Furthermore, the ROKIT Locator uses SI units everywhere unless explicitly and clearly stated otherwise. The application data sent or received by the ROKIT Locator uses little-endian byte order for integer values. Floating point values are encoded in IEEE-754 data types, unless noted otherwise.

Rexroth also provides a ROS and ROS2 interface to the Rexroth ROKIT Locator (with limited functionality) free of charge on Github: https://github.com/boschglobal/locator_ros_bridge. It translates ROS messages to the ROKIT Locator API and vice versa. It also allows to control the ROKIT Locator via ROS service calls. Therefore, the Apache License (Version 2.0) applies and Rexroth does not assume any further warranty or liability with respect to the ROS interface.

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 ROKIT Locator or request a response containing specific information. For each query received, the ROKIT Locator 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 unless the asynchronous features are used as described in section 3.1.9.

The “id” JSON object can be chosen freely but it must only consist of printable ASCII characters. A list of the currently running JSON Requests can be received using the SystemJsonRequestIdList Interface.

Some query messages may have optional entries. These entries are marked as optional within the message specification given in this document. If an optional entry is omitted during a request, the message will be interpreted as if the missing entry had been set to its default value. The default value for each optional entry is also given within the message description in this document. We recommend to always specify all entries in a query message in full and to only rely on the default values for forward compatibility when new, optional entries are added.

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 ROKIT Locator 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.

3.1.4 Input validation

As a security feature input validation is activated. If the content of the message does not conform to the message specification, the message will be rejected. In this case, the ROKIT Locator responds with the error code -32602 (INVALID_PARAMS). Messages sent by the ROKIT Locator in response to a user request are also validated in the same way. If the ROKIT Locator attempts to send a message that fails validation, the user will instead receive the error code -32603 (INTERNAL_ERROR). This ensures that the user only receives valid messages.

Due to technical reasons, not all inputs can be validated. In particular, the limits of numeric types such as Integer64 or NonnegativeInteger64 cannot be checked if the limit itself exceeds the values that can be represented by the underlying data type. In this case, the user himself must take care not to send values that exceed these limits. Otherwise, undefined behavior may result.

3.1.5 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. Unlike query messages, response messages never have optional entries and are always sent in full.

3.1.6 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 ROKIT Locator or ROKIT Locator Support 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 450 Session IDs is allowed before getting the response code `SESSION_MAXIMUM_NUMBER_REACHED`, as defined in Session Specific ResponseCodes. When reaching the session limit, the user is still able to start 50 additional short-term sessions. A short-term session is a session with an expiration timeout less than or equal to 60 seconds. This

behavior allows to log in and e.g. manually revoke active sessions even when the session limit is reached.

3.1.7 Revocation IDs

In certain situations it can be necessary to revoke a session from another user. In order to do so a revocation id can be used which, in contrast to the session id, can be requested by any admin user using an API call. The revocation id can only be used to revoke a session.

3.1.8 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:`.

3.1.9 Optional: Asynchronous RPC Calls and Responses

The TCP connection, i.e., the transport layer of the HTTP protocol, is prone to fail in cases of unstable wireless links. To handle these cases without waiting for an response that never comes, clients need to use request timeouts. However if the RPC call has a very long processing time, such timeouts can lead to premature disconnects. Even if the timeout is well chosen, it is possible that the TCP connection is actually broken. The response in such a case is definitely lost.

To allow the user to handle such cases more explicitly, any RPC call can be used asynchronously effectively separating the call to request the remote method from the one fetching the method's response. This is done by using the optional `params` member `queryMode`. The initial response to an asynchronous request will contain a `secret` field (type string). The user can use this `secret` to get the response of the asynchronous request at a later point in time, even if the original TCP connection has been closed in the meantime.

Query Modes

Label	Value	Description
BLOCKING_REQUEST	0	(default) The call is processed and the `result` field of the answer contains the methods `response`
ASYNC_REQUEST	1	The call is processed and the `result` field of the answer contains a shared `secret` to fetch the response later on. If the server is busy, the response can still be delayed
BLOCKING_GET	2	The `params` field must contain the shared `secret` for which the response is queried. The call returns only after the method is finished
ASYNC_GET	3	The `params` field must contain the shared `secret` for which the response is queried. The call returns immediately even if the result is not ready. If the server is busy, the response can still be delayed

Label	Value	Description
DISCARD	4	The `params` field must contain the shared `secret` for which the procedure should be stopped (if possible) and the result discarded. Note that most methods can not be stopped and this call will only discard the result

Additional JSON-RPC error codes In the optional asynchronous mode, the following additional error codes are used.

Label	Value	Description
WOULD_BLOCK	-32000	The start of the query would be blocking
SECRET_INVALID	-32001	The provided `secret` is invalid
NOT_READY	-32002	The result associated with this `secret` is not ready
SECRET_DISCARDED	-32003	The provided `secret` was valid but the method was discarded
SECRET_RETIRED	-32004	The provided `secret` was valid but retired after the result was fetched

Request Call The typical asynchronous RPC request to start the method will look like this:

```
...
...
Content-Type: application/json; charset=utf-8
{
  "id": <ID>,
  "jsonrpc": "2.0",
  "method": "someQueryMethod",
  "params": {
    "query": {
      QueryType: see message types
    },
    "queryMode": 1
  }
}
```

Request Response The typical response to an asynchronous RPC response will look like this:

```
...
...
Content-Type: application/json; charset=utf-8
{
  "id": <ID>,
  "jsonrpc": "2.0",
  "result": {
    "secret": "<shared secret>"
```

```
  }
}
```

The shared secret can later on be used to fetch the response. On failure, a error object will be returned. The WOULD_BLOCK error code defined above is returned if the method is already processing from an prior call.

Get Call The typical asynchronous RPC request to get the result of a method will look like this:

```
...
...
Content-Type: application/json; charset=utf-8
{
  "id": <ID>,
  "jsonrpc": "2.0",
  "method": "someQueryMethod",
  "params": {
    "secret": "<shared secret>"
    "queryMode": 2 or 3
  }
}
```

Every other get call that is still in waiting (all blocking calls) will return with an error object containing the NOT_READY error code defined above. Thus, only one get call is valid at each time to avoid that the answer is send to a retired connection.

Get Response If the call was successful, i.e., the result was fetched, the response looks exactly as defined in the blocking case (see Response section). If the result is not ready, an error object will be returned containing the NOT_READY error code defined above.

Discard Call The typical asynchronous RPC request to discard a previous query will look like this:

```
...
...
Content-Type: application/json; charset=utf-8
{
  "id": <ID>,
  "jsonrpc": "2.0",
  "method": "someQueryMethod",
  "params": {
    "secret": "<shared secret>"
    "queryMode": 4
  }
}
```

Every call to get the result will be unblocked immediately. Since the secret is invalid after discard is done, all other get calls that are still in waiting (all blocking calls) will return with an error object containing the SECRET_DISCARDED error code defined above.

Discard Response If the call was successful, i.e., the method was still processing or the result was not yet fetched, the now discarded and further invalid secret is returned. On failure, a error object will be returned.

Discarding of lingering results If a new method is started after a previous call to the method is finished but before the result is fetched, the lingering result is discarded and the secret is listed as discarded from there on. This can occur either by sending any query call after the method finished but before a get call was received or by sending a blocking query call while the method is still running. If a blocking get call is waiting for the result, it will always be able to fetch it before it is discarded, and the secret is listed as retired.

Unblocking get calls Only the last of multiple get calls with the same secret and method is considered, every previous get call will return with the NOT_READY error code defined above

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. For the binary interface there is only a single protocol supported, the Transmission Control Protocol[1] Internet Protocol (TCP/IP).

3.2.1 TCP/IP Protocol

In practice, the Binary Interfaces with TCP/IP protocol use a simple unidirectional push pattern:

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 ROKIT Locator by acting as a server; after connecting to such a socket, the user will receive Datagrams as soon as available.
7. Push consumers receive data for the ROKIT Locator 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.

[1] RFC 9293: https://datatracker.ietf.org/doc/html/rfc9293

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.

Unless explicitly allowed in the specification of a given interface, the user must not send any data to a push provider. If the user sends data to a push provider which should not receive data, the Binary Interface connection will be closed.

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.

3.2.2 TCP/IP via Binary Interface Proxy

If desired, all ROKIT Locator TCP/IP Binary Interfaces may be accessed through a single TCP port. This is achieved through a Binary Interface Proxy, which forwards communications to all Binary Interfaces from a single TCP port. Using this feature can greatly reduce the number of open TCP ports used by the ROKIT Locator.

When using the Binary Interface Proxy, the user no longer connects directly to the individual TCP ports of each Binary Interface. Instead, the user should connect to the single TCP port of the Binary Interface Proxy, as given in the list of ports.

After the connection to the Binary Interface Proxy has been established, the user must specify the Binary Interface that the user wants to connect to. This is done by sending the following datagram via the TCP connection that was established to the Binary Interface Proxy.

Name	Size (B)	Type	Restriction
product	2	UnsignedInteger16	Should be 0.
port	2	UnsignedInteger16	Must be > 0.

The `product` is reserved for future use and should be set to 0. The `port` is the port of the Binary Interface to which the user wishes to connect, as given in the list of ports.

After this datagram has been sent, the TCP connection will behave as if the user had connected directly to the Binary Interface in question. This applies to both the TLS and non-TLS ports of each Binary Interface. When connecting to the TLS port of a Binary Interface, the TLS handshake must therefore only be performed after the above datagram has been sent. This results in the following steps:

1. Connect to the Binary Interface Proxy via the given TCP port.
2. Send the `product` and `port` as described above.
3. If `port` is a TLS port: Perform the TLS handshake as usual.
4. The connection to the Binary Interface is now fully established.

If the `port` sent to the Binary Interface Proxy is not valid for the ROKIT Locator or if Binary Interface Proxy is not configured to handle this `port`, the Binary Interface Proxy will immediately close the TCP connection to the user.

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.

3.2.4 Datagram extensions

Since the Binary Interface does not provide bidirectional communication, it depends completely on well defined datagrams to ensure uncorrupted data deserialization. Thus, the interface will not be automatically forward compatible and user implementations have to be adapted if the datagram changes.

To mitigate this limitation, any datagram may contain an extension area that provides optional content for future expansions to the datagram. This content can be skipped completely or partially without fully deserializing the extension payload. It is recommended to implement this behavior in a forward compatible manner to ensure that minor API version changes do not break compatibility between the ROKIT Locator and user-written software.

4 Client Control Mode

The ROKIT Locator offers numerous functionalities, such as recording laser scan, building a map, or localization. To implement these diverse functionalities within a single product element, the ROKIT Locator Client has **7** 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 ROKIT Locator Client localizes itself to a map received from the ROKIT Locator 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.
- **LASEROUTPUT**: This mode is *RUN*-state while the user is retrieving data from the laser scanners, for example to create a mask for the laser scans.
- **EXPANDMAP**: This mode is *RUN*-state while the map expansion is activated for overwriting or expanding the prior map. This mode takes only effect in combination with the modes VISUALRECORDING or MAP.

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 or an error occurs, 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. Additionally, the LOC state cannot RUN in parallel with the VISUALRECORDING state and vice versa.

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 ROKIT Locator Client's RPC interface. These methods are documented in the chapter ROKIT Locator Client – RPC Methods. Note that client control modes may only be changed from RUN to READY or READY to RUN. Any other mode change, including changing from RUN to RUN or READY to READY, will result in a `NOT_IN_REQUIRED_STATE` error. Beyond this error, attempting such a mode change has no other negative effects.

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 ROKIT Locator Client is actually in the correct mode.

Note that the modes MAP, REC, and VISUALRECORDING may change from the RUN to the READY state without any user interaction. For the MAP mode, this occurs when the ROKIT Locator Client has finished building a map. The ROKIT Locator Client may leave the RUN state of the REC and VISUALRECORDING modes in case of an error. This may occur if the connection to a laser scanner is lost during an active recording. In all of these cases, the reason for the mode change will be reported to the customer through the diagnostic module. Note that the EXPANDMAP mode will remain unchanged if the MAP or VISUALRECORDING mode changes automatically.

Also note that while any of the client control modes is in RUN state, a served license cannot be returned using the RPC licensingFeatureReturnServedLicense.

5 Response Codes

5.1 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.

5.2 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	Licensing
0x0004	Config
0x0005	AboutBuild
0x0006	Certificate
0x0007	System
0x0008	Diagnostic
0x0100	ClientRecording
0x0101	ClientMap
0x0102	ClientLocalization
0x0103	ClientManualAlign
0x0104	ClientGlobalAlign
0x0105	ClientLaserMask
0x0106	ClientUser

Identifier (16 bit)	Module
0x0107	ClientExpandMap
0x0108	ClientSensor
0x0200	ServerMap
0x0201	ServerUser
0x0202	ServerPostAlign
0x0300	SupportReport
0x0301	SupportRecovery
0x0302	SupportFleetinfo
0x04XX	reserved for internal use
0x05XX	reserved for internal use

5.3 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 A user tries to operate the ROKIT Locator 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 ROKIT Locator after their session has expired due to inactivity.

Response code (64 bit, hexadecimal) RESPONSE	**Description** **Example**
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 ROKIT Locator Client is already busy recording sensor data.
0x0000 0000 0000 0008 FEATURE_NOT_LICENSED	The ROKIT Locator is not licensed to perform the desired operation. A user tries to build a map on a ROKIT Locator 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. The ROKIT Locator 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 ROKIT Locator. A user tried to delete a recording from which the ROKIT Locator is currently building a map.

5.4 *SESSION*-SPECIFIC RESPONSE CODES

Response code (64 bit, hexadecimal)	**Description**
RESPONSE	**Example**
0x0002 0000 0000 0001	The given user credentials were not valid.
SESSION_INVALID_CREDENTIALS	A user tried to log in with an incorrect password or username.
0x0002 0000 0000 0003	The number of open sessions has reached the predetermined maximum.
SESSION_MAXIMUM_NUMBER_REACHED	A user tried to log in while too many existing sessions were already active.
0x0002 0000 0000 0004	Login functionality disabled temporarily due to too many failed attempts.
SESSION_TOO_MANY_FAILED_ATTEMPTS	A user made too many login attempts with invalid credentials within a short timespan.
0x0002 0000 0000 0005	There is no session associated with the provided revokation id.
SESSION_WITH_REVOKE_ID_NOT_FOUND	A user provided a wrong revokation id.
0x0002 0000 0000 0006	The session associated with the provided revokation id is already expired. It can not be revoked.
SESSION_WITH_REVOKE_ID_EXPIRED	A user provided a revokation id for an already expired session.

5.5 *LICENSING*-SPECIFIC RESPONSE CODES

Response code (64 bit, hexadecimal)	**Description**
RESPONSE	**Example**
0x0003 0000 0000 0001	The ROKIT Locator has not received any valid license files.
LICENSING_NO_VALID_LICENSE	A user attempts to retrieve license information without first sending a license.
0x0003 0000 0000 0002	The license has expired.
LICENSING_LICENSE_EXPIRED	The date of expiration has passed while using a non-permanent license.
0x0003 0000 0000 0003	The method is not available for the currently selected licensing method.
LICENSING_METHOD_NOT_AVAILABLE	A user calls a licensing API method that is not applicable with the selected licensing method.

Response code (64 bit, hexadecimal) RESPONSE	Description Example
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 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 preferred 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. 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.

Response code (64 bit, hexadecimal) RESPONSE	**Description** **Example**
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.
0x0003 0000 0002 0005 LICENSING_TPM_NO_ROOT	TPM licensing not available due to missing superuser privileges. The container needs to be started with superuser privileges if TPM licensing should be used.
0x0003 0000 0003 0000 LICENSING_LOCALSERVER_URL_INVALID	The url of the Revenera License Server is invalid. The config Licensing.localServerUrl is not set.
0x0003 0000 0003 0001 LICENSING_LOCALSERVER_REQUEST_ERROR	The request for a served license has an error. The connection to the Revenera License Server is bad.
0x0003 0000 0003 0002 LICENSING_LOCALSERVER_RESPONSE_ERROR	The response from the Revenera License Server has an error. The response from the Revenera License Server does not contain a valid license.
0x0003 0000 0003 0003 LICENSING_LOCALSERVER_HOSTID_ERROR	The ROKIT Locator reported an error while retrieving the host id.
0x0003 0000 0003 0004 LICENSING_LOCALSERVER_RETURN_ERROR	The ROKIT Locator reported an error while trying to return the license.
0x0003 0000 0003 0005 LICENSING_LOCALSERVER _NOT_ALL_FEATURES	The ROKIT Locator did not receive all requested license features from the Revenera License Server
0x0003 0000 0003 0006 LICENSING_LOCALSERVER _API_NOT_ALLOWED_IN_AUTOMATIC_MODE	The ROKIT Locator rejected the API call in automatic mode for the Revenera License Server The config Licensing.getServedLicenseAutomatically is set to true.
0x0003 0000 0003 0007 LICENSING_LOCALSERVER _NO_LICENSE_TO_RETURN	The ROKIT Locator rejected the API call since there is no existing license to return. The ROKIT Locator has been factory reset and has no license to return.

Response code (64 bit, hexadecimal) RESPONSE	Description Example
0x0003 0000 0004 0000 LICENSING_IDFILE_NOT_FOUND	The signed licensing ID file was not found or could not be read. The user attempts to use ID file-based licensing but did not enroll a licensing ID file.
0x0003 0000 0004 0001 LICENSING_IDFILE_VERIFY_FAILED	The cryptographic signature of the licensing ID file failed verification. The user attempts to use ID file-based licensing but the licensing ID file was corrupted.
0x0003 0000 0005 0000 LICENSING_CTRLX_NOT_FOUND	Could not connect to the ctrlX OS to enable the licensing. The user attempts to use ctrlX licensing with docker in a non-ctrlX environment.

5.6 *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 or the user does not have sufficient authorization to change a given parameter. 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.
0x0004 0000 0000 0003 CONFIG_DUPLICATE_PARAMETER_KEY	A given configuration key was set more than once in a single call. A user called configSet and specified a key, e.g. `ClientSensor.laser.type`, more than once.

5.7 *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.

5.8 *SUPPORTREPORT*-SPECIFIC RESPONSE CODES

Response code (64 bit, hexadecimal) RESPONSE	Description Example
0x0300 0000 0000 0001 SUPPORTREPORT_TIMESPAN_INVALID	The timespan for the requested support report was invalid. The requested start or end time lies in the future.

5.9 *SUPPORTRECOVERY*-SPECIFIC RESPONSE CODES

Response code (64 bit, hexadecimal) RESPONSE	Description Example
0x0301 0000 0000 0001 SUPPORTRECOVERY_VERSION_INCOMPATIBLE	The recovery point has an incompatible version while trying to restore from this recovery point.
0x0301 0000 0000 0002 SUPPORTRECOVERY_READ_ARCHIVE_FAILED	The ROKIT Locator failed to extract or decrypt information from the recovery point.
0x0301 0000 0000 0003 SUPPORTRECOVERY_RECOVERY_FAILED	The ROKIT Locator failed to recover the data from the recovery point.
0x0301 0001 0000 0001 SUPPORTRECOVERY_ARCHIVE_INVALID	The given archive was invalid. The archive has been modified or damaged after it was exported.
0x0301 0001 0000 0002 SUPPORTRECOVERY_ARCHIVE_WRONG_TYPE	The given archive does not contain the correct type of entity. The archive contains a Client recovery point where a Server recovery point was expected.
0x0301 0001 0000 0003 SUPPORTRECOVERY_ARCHIVE_INCOMPATIBLE	The given archive was created by a component of an incompatible version. The archive was exported by a ROKIT Locator Server Support with a higher (newer) version than that of the server being used.

5.10 *CLIENTRECORDING*-SPECIFIC RESPONSE CODES

Response code (64 bit, hexadecimal) RESPONSE	Description Example
0x0100 0001 0000 0001 CLIENTRECORDING_ARCHIVE_INVALID	The given archive was invalid. The archive has been modified or damaged after it was exported.
0x0100 0001 0000 0002 CLIENTRECORDING_ARCHIVE_WRONG_TYPE	The given archive does not contain the correct type of entity. The archive contains a Server Map where a Recording was expected.
0x0100 0001 0000 0003 CLIENTRECORDING_ARCHIVE_INCOMPATIBLE	The given archive was created by a ROKIT Locator Client of an incompatible version. The archive was exported by a ROKIT Locator Client with a higher (newer) version than that of the client being used.

Response code (64 bit, hexadecimal) RESPONSE	Description Example
0x0100 0001 0000 0004 CLIENTRECORDING_NOT_ENOUGH_SPACE	The ROKIT Locator Client does not have enough free storage space to safely start a new (visual) recording.

5.11 *CLIENTMAP*-SPECIFIC RESPONSE CODES

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

5.12 *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

5.13 *CLIENTEXPANDMAP*-SPECIFIC RESPONSE CODES

Response code (64 bit, hexadecimal) RESPONSE	Description Example
0x0107 0000 0000 0001 CLIENTEXPANDMAP_OVERWRITEZONE_NOT_FOUND	An overwrite area with the specified Id could not be found. A user tried to delete an overwrite area that does not exist
0x0107 0000 0000 0002 CLIENTEXPANDMAP_LOAD_PRIOR_MAP_FAILED	An overwrite area with the specified Id could not be found. A user tried to delete an overwrite area that does not exist
0x0107 0000 0000 0003 CLIENTEXPANDMAP_PRIOR_MAP_PATH_INVALID	The recording with the prior map has an invalid prior map path. A user tried to enable the map expansion with an invalid prior map path.
0x0107 0000 0000 0004 CLIENTEXPANDMAP_MULTI_PRIOR_MAP_FOUND	The recording contains more than one prior map. A user tried to extend a map with a recording which contains unexpectedly multiple prior maps.
0x0107 0000 0000 0005 CLIENTEXPANDMAP_NO_PRIOR_MAP_FOUND	The recording does not contain any prior map. A user tried to extend a map with a recording without prior map inside the recording.
0x0107 0000 0000 0006 CLIENTEXPANDMAP_OVERWRITEZONE_INVALID	The overwrite area is invalid. A user tried to add an overwrite area that has less than 3 points or overlapping edges.

5.14 *CLIENTSENSOR*-SPECIFIC RESPONSE CODES

Response code (64 bit, hexadecimal) RESPONSE	Description Example
0x0108 0000 0000 0001 CLIENTSENSOR_CALIBRATION_FAILED	Calibration parameters could not be derived from the given data. The data did not contain enough sensor data to perform an accurate calibration.

Response code (64 bit, hexadecimal) RESPONSE	Description Example
0x0108 0000 0000 0002 CLIENTSENSOR_CALIBRATION_NO_VALID_LASER2_DATA	Calibration parameters could not be derived from the given data. The data did not contain enough usable data of the secondary laser.
0x0108 0000 0000 0003 CLIENTSENSOR_CALIBRATION_NO_VALID_ODOMETRY_DATA	Calibration parameters could not be derived from the given data. The data did not contain enough usable data of the odometry sensor.

5.15 *SERVERMAP*-SPECIFIC RESPONSE CODES

Response code (64 bit, hexadecimal) RESPONSE	Description Example
0x0200 0000 0000 0001 SERVERMAP_ZONE_NOT_FOUND	No zone with the given id was found in the map with the given name. No zone with given id exists in this map.
0x0200 0000 0000 0002 SERVERMAP_ZONE_INVALID	The given zone was invalid. The zone's polygon has self intersections or does not form a two dimensional object.
0x0200 0001 0000 0001 SERVERMAP_ARCHIVE_INVALID	The given archive was invalid. The archive has been modified or damaged after it was exported.
0x0200 0001 0000 0002 SERVERMAP_ARCHIVE_WRONG_TYPE	The given archive does not contain the correct type of entity. The archive contains a Recording where a Server Map was expected.
0x0200 0001 0000 0003 SERVERMAP_ARCHIVE_INCOMPATIBLE	The given archive was created by a ROKIT Locator Server of an incompatible version. The archive was exported by a ROKIT Locator Server with a higher (newer) version than that of the server being used.

5.16 *SERVERPOSTALIGN*-SPECIFIC RESPONSE CODES

Response code (64 bit, hexadecimal) RESPONSE	**Description** **Example**
0x0202 0000 0000 0001 SERVERPOSTALIGN_ALIGNMENT_FAILED	The calculations to align the map failed. The requested alignment was impossible to fulfill due to inconsistent scan coordinates.
0x0202 0000 0000 0002 SERVERPOSTALIGN_MAP_INCOMPATIBLE	The chosen server map is not suitable for alignment. The specified map was sent to the server using ROKIT Locator Client 1.5 or older.
0x0202 0000 0000 0003 SERVERPOSTALIGN_DUPLICATE_SCAN_ID	The alignment request contained multiple entries with the same scan ID.
0x0202 0000 0000 0004 SERVERPOSTALIGN_INVALID_SCAN_ID	The alignment request contained a scan ID that does not exist in the chosen map.

6 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.

6.1 TYPES

6.1.1 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.

6.1.2 Base64EncodedString (string)

A Base64 encoded character string stores data as string that contains only characters that are in RFC4648-compliant Base64 format. There are various tools for encoding data in the Base64 format, see FAQ for an example.

6.1.3 EmptyString (string)

A string with a fixed length of 0.

6.1.4 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].

6.1.5 NonnegativeInteger64 (integer)

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

6.1.6 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 * 10^{308}, 1.797693134862 * 10^{308}]. This data type does not accept special values allowed under IEEE754, such as NaN or Infinity.

6.1.7 ResponseCode (NonnegativeInteger64)

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

6.1.8 UUID (string)

UUIDs are RFC4122-compliant Universally Unique IDentifiers. 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.

6.1.9 SessionId (string)

A Universal Unique IDentifier UUID, as described in the relevant section.

6.1.10 RevokeId (string)

A Universal Unique IDentifier UUID, as described in the relevant section.

6.1.11 Username (AsciiCharacterString, length > 0)

A username is a non-empty AsciiCharacterString of up to 121 characters.

6.1.12 Password (AsciiCharacterString, length > 0)

A password is a non-emtpty AsciiCharacterString of up to 511 characters.

6.1.13 Size (NonnegativeInteger64)

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

6.1.14 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.

6.1.15 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.

6.1.16 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.

6.1.17 DataExchangeArchiveName (AsciiCharacterString, length > 0)

Archive files used for data storage and exchange 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 archive names.

6.1.18 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.

6.1.19 HttpsUrl (AsciiCharacterString, length > 0)

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

6.1.20 BasicAuth (AsciiCharacterString, length > 0)

A non-empty string starting with a case insensitive word ”basic” followed by a space and the base64-encoded username and password as defined in rfc7617.

6.1.21 BearerToken (AsciiCharacterString, length > 0)

A non-empty string starting with a case insensitive word ”bearer” followed by a space and the bearer token as defined in rfc6750.

6.1.22 Port (integer)

A non-empty integer representing a valid port. The interval of valid ports is [1, 65535].

6.1.23 IPv4Address (string)

A numeric IPv4 address and TCP/UDP port in the format *A.B.C.D:P*. Here, *A, B, C, D* are decimal string representations of positive integers with up to three digits. *P* is the decimal string representation of a positive integer with up to five digits. Note that the overall string must be a valid IPv4 address and TCP or UDP port, such as 192.168.0.1:6060. As such, *A, B, C, D* must not exceed 255, while *P* must not exceed 65535.

6.2 OBJECTS

6.2.1 Timestamp

- “valid”: boolean
- “time”: Integer64 (unit depends on given resolution). “time” multiplied with “resolution” divided by 1000000000 must lie within the limits of an Integer64, resulting in the following additional limits:
  - If “resolution” is 1: -9223372036 $\leq$ “time” $\leq$ 9223372036,
  - if “resolution” is 100: -922337203685 $\leq$ “time” $\leq$ 922337203685,
  - if “resolution” is 1000: -9223372036854 $\leq$ “time” $\leq$ 9223372036854,
  - if “resolution” is 1000000: -9223372036854775 $\leq$ “time” $\leq$ 9223372036854775.
- “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.

6.2.2 TimeInterval

- “valid” : boolean
- “time” : Integer64 (unit depends on given resolution). “time” multiplied with “resolution” divided by 1000000000 must lie within the limits of an Integer64, resulting in the following additional limits:
  - If “resolution” is 1: -9223372036 $\leq$ “time” $\leq$ 9223372036,
  - if “resolution” is 100: -922337203685 $\leq$ “time” $\leq$ 922337203685,
  - if “resolution” is 1000: -9223372036854 $\leq$ “time” $\leq$ 9223372036854,

  - if “resolution” is 1000000: -9223372036854775 $\leq$ “time” $\leq$ 9223372036854775.
- “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.

6.2.3 Point2D

- “x”: IEEE754Double (in meters)
- “y”: IEEE754Double (in meters)

Describes the point of an object within a 2D coordinate system, given by the object position “x”, “y” (in meters) without orientation.

6.2.4 Pose3D

- “x”: IEEE754Double (in meters)
- “y”: IEEE754Double (in meters)
- “z”: IEEE754Double (in meters)
- “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” (in meters) and the orientation quaternion “qw”, “qx”, “qy”, and “qz”.

6.2.5 Transform3D

- “x”: IEEE754Double (in meters)
- “y”: IEEE754Double (in meters)
- “z”: IEEE754Double (in meters)
- “qw”: IEEE754Double
- “qx”: IEEE754Double
- “qy”: IEEE754Double
- “qz”: IEEE754Double

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

6.2.6 Pose2D

- “x”: IEEE754Double (in meters)
- “y”: IEEE754Double (in meters)
- “a”: IEEE754Double (in radians)

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

6.2.7 Transform2D

- “x”: IEEE754Double (in meters)
- “y”: IEEE754Double (in meters)
- “a”: IEEE754Double (in radians)

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

6.2.8 PointCluster

- “prototype”: Pose2D
- “fromIndex”: Size
- “toIndex”: Size

A PointCluster represents a cluster of points within a separately transmitted point cloud. A cluster of points is characterized by its position (in meters) and orientation (in radians). This position and orientation is described as a prototype of type Pose2D. For example, a cluster that contains points that belong to a physical object like a crate could be characterized by the position of the objects center and the orientation of one of its faces.

Figure 1: Example of a PointCluster object. The point cloud given contains a cluster that consists of the points with indices within [6,8]–shown in blue within the image. The prototype of the cluster is represented by a blue arrow. The appropriate closed range of indices would be [6,8], i.e., fromIndex = 6, toIndex = 8.

The cluster consists of all the points within the point cloud whose index lies within a given range [fromIndex, toIndex], as depicted in Figure 1. Therefore, only consecutive points can be part of a cluster and the point cloud may be reordered to achieve this.

6.2.9 Container

- “contentEncoding”: “base64” (string)
- “contentMediaType”: AsciiCharacterString (length > 0)
- “content”: Base64EncodedString 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 an 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. Note: Specific modules may specify additional media types not listed here.

Media Type “application/PointCloud2D” A container of this type holds a 2D point cloud as a base64-encoded (see FAQ for an 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) (in meters).

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

7 Common RPC Methods

The RPC methods listed in this section are common to various Rexroth ROKIT products and components (if product consists of multiple parts). Therefore, they are listed separately in this chapter. Please be aware that different products may have different versions of this common API modules implemented and therefore the API may be different for them.

7.1 API MODULE INFORMATION (MODULE: *ABOUTMODULES*)

7.1.1 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

7.1.2 aboutModulesComponent

This call is used to query the component identifier.

- QueryType: AboutModulesQueryMessage
- ResponseType: AboutModulesComponentResponseMessage

Query Value None (empty JSON object)

Response Value

- response code as defined in the relevant section
- element of AboutModulesComponent, containing the corresponding component type

7.1.3 Messages

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

AboutModulesListResponseMessage

- “responseCode”: ResponseCode
- “modules”: array of AboutModulesModule

AboutModulesComponentResponseMessage

- “responseCode”: ResponseCode
- “component”: AboutModulesComponent

7.1.4 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.

AboutModulesComponent

- “componentType”: AboutModulesComponentType

7.1.5 Types

AboutModulesComponentType (integer)

Label	Value	Description
UNDEFINED	0	Undefined component, should not occur
LOCATOR_CLIENT	1	The component ROKIT Locator Client
LOCATOR_SERVER	2	The component ROKIT Locator Server
LOCATOR_CLIENT_SUPPORT	3	The component ROKIT Locator Client Support
LOCATOR_SERVER_SUPPORT	4	The component ROKIT Locator Server Support

7.2 SESSION CONTROL (MODULE: *SESSION*)

7.2.1 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 less or with 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.

7.2.2 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

7.2.3 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

7.2.4 sessionGroupInfo

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

- QueryType: SessionQueryMessage
- ResponseType: SessionGroupsResponseMessage

Query Value

- sessionId: the session requesting its user groups

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)

7.2.5 sessionList

Requests a list of session info objects for non-expired and expired (limited number) sessions.

- QueryType: SessionQueryMessage
- ResponseType: SessionListResponseMessage

Query Value

- sessionId: the session requesting the current session list

Response Value

- response code as defined in the relevant section
- array of session info objects

7.2.6 sessionRevoke

Revokes a session with a revocation id.

- QueryType: SessionRevokeMessage
- ResponseType: SessionResponseMessage

Query Value

- sessionId: the session requesting the revocation
- revokeId: the revocation id for the respective session which shall be revoked

Response Value

- response code as defined in the relevant section

7.2.7 sessionRevokeAll

Revokes all sessions including the one where this method is being called from.

- QueryType: SessionQueryMessage
- ResponseType: SessionResponseMessage

Query Value

- sessionId: the session requesting the revocation

Response Value

- response code as defined in the relevant section

7.2.8 sessionRevokeId

Returns the revocation id of the calling session.

- QueryType: SessionQueryMessage
- ResponseType: SessionRevokeIdResponseMessage

Query Value

- sessionId: the session for which the revocation id is being requested

Response Value

- response code as defined in the relevant section
- revocation id for the provided session id

7.2.9 Messages

SessionLoginMessage

- “timeout”: LimitedMax30DaysTimeInterval
- “userName”: Username
- “password”: Password

SessionLoginResponseMessage

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

SessionQueryMessage

- “sessionId”: SessionId

SessionResponseMessage

- “responseCode”: ResponseCode

SessionGroupsResponseMessage

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

SessionListResponseMessage

- “responseCode”: ResponseCode
- “sessions”: array of SessionInfo

SessionRevokeMessage

- “sessionId”: SessionId
- “revokeId”: RevokeId

SessionRevokeIdResponseMessage

- “responseCode”: ResponseCode
- “revokeId”: RevokeId or EmptyString

7.2.10 Objects

SessionInfo

- “lastActivity”: Timestamp
- “expiration”: Timestamp
- “isExpired”: boolean
- “revokeId”: RevokeId

LimitedMax30DaysTimeInterval TimeInterval with maximum 30 days

- If “resolution” is 1: “time” $\leq$ 2592000,
- if “resolution” is 100: “time” $\leq$ 259200000,
- if “resolution” is 1000: “time” $\leq$ 2592000000,
- if “resolution” is 1000000: “time” $\leq$ 2592000000000.

7.3 DIAGNOSTIC INFORMATION (MODULE: *DIAGNOSTIC*)

7.3.1 diagnosticList

Retrieves available diagnostic entries from the buffer.

Optional filters can be used to limit the entries that are retrieved. These filters consist of three different types: severity, group, and time filters. If at least one filter of a given type is specified, the response will only contain entries that match at least one of the filters for that type. Filters of the same type are therefore combined with a logical OR, while the different types of filters are combined with a logical AND. If no filter of a given type is specified, that filter type is ignored and will not affect the response. If no filters are specified at all, the ROKIT Locator returns all available diagnostic entries.

- QueryType: DiagnosticFilterMessage
- ResponseType: DiagnosticArrayResponseMessage

Query Value

- sessionId: the session requesting diagnostic information
- filters (optional, default value: no filters): filters which limit the diagnostic entries that are returned

Response Value

- “responseCode”: ResponseCode
- array of all diagnostic messages available as per the permission of the given session and the specified filters (if any)

7.3.2 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

7.3.3 Messages

DiagnosticQueryMessage

- “sessionId”: SessionId

DiagnosticResponseMessage

- “responseCode”: ResponseCode

DiagnosticFilterMessage

- “sessionId”: SessionId
- “filters”: DiagnosticFilter (optional)

DiagnosticArrayResponseMessage

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

7.3.4 Objects

DiagnosticFilter Each of the following filter arrays is optional.

- “severityFilter”: array of DiagnosticSeverity
- “groupFilter”: array of DiagnosticGroup
- “timeFilter”: array of DiagnosticAbsoluteTimeFilter and DiagnosticRelativeTimeFilter

DiagnosticAbsoluteTimeFilter

- “timestampStart”: Timestamp
- “timestampEnd”: Timestamp

Adding a filter of this type will include all diagnostic messages with a timestamp that lies between “timestampStart” and “timestampEnd”. This is an inclusive interval.

If the “valid” field of “timestampEnd” is “false”, the time interval has no upper limit. Conversely, if the “valid” field of “timestampStart” is “false”, this time interval has no lower limit.

DiagnosticRelativeTimeFilter

- “timestamp”: Timestamp
- “timeInterval”: TimeInterval

Adding a filter of this type will include all diagnostic messages for which the timestamp lies within a duration “timeInterval” from the “timestamp”. A positive “timeInterval” will filter for messages with a timestamp that lies after “timestamp”, whereas a negative “timeInterval” will filter for messages with a timestamp that lies before the given “timestamp”. This is an inclusive interval.

If the “valid” field of “timestamp” is “false”, then the current time is used as the “timestamp”. If the “valid” field of “timeInterval” is “false”, the “timeInterval“ is considered to be 0.

DiagnosticEntry 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.

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

Please consider the following example.

- diagnosticText: `'New session {sessionId} for a member of the group {group}'`
- additionalInfo: `['group','admin','sessionId','4f518']`
- assembled message: `New session 4f518 for a member of the group admin`

7.3.5 Types

DiagnosticGroup (integer) The DiagnosticGroup enumeration can be used to filter the DiagnosticEntries for specific groups. Refer to the ROKIT Locator manual for the meaning of these entry types.

Group	Value
Undefined	0
SystemEvent	1
APICall	2
DataInput	3
DataOutput	4

DiagnosticCode (integer) 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 severity level of entry according to the section DiagnosticSeverity.

DiagnosticSeverity (integer) The three most significant bits in DiagnosticCode specify one of four severity levels of diagnostic entries. Refer to the ROKIT Locator manual for the meaning of these codes.

Severity	Value as integer	Bits 63-61
INFO	6	110
NOTICE	5	101
WARNING	4	100
ERR	3	011
CRIT	2	010

7.4 SOFTWARE LICENSING (MODULE: *LICENSING*)

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. Chapter 15.1 lists the available methods for host machine identification. After acquiring a license for a given host the associated license file must be sent to the product element using the licensingFeatureSet method.

This section merely describes the API functionality used to license the Rexroth ROKIT Locator. Refer to the Licensing section for information on what licenses are required to use the other functionality described in this document.

Set the licensing configuration parameters (chapter 12) correctly, before using the following API methods.

7.4.1 licensingFeatureGetTrustedPlatformModuleInformation

Retrieves the Trusted Platform Module Information of the host. This method is not available if the `Licensing.licensingMethod` config parameter is set to `CTRLX`. If `CTRLX` is set, attempting to call this method will result in a `LICENSING_METHOD_NOT_AVAILABLE` error.

- 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

7.4.2 licensingFeatureGetHostId

Retrieves the hardware-specific Host ID. This method is not available if the `Licensing.licensingMethod` config parameter is set to `CTRLX`. If `CTRLX` is set, attempting to call this method will result in a `LICENSING_METHOD_NOT_AVAILABLE` error.

- 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

7.4.3 licensingFeatureSet

Sends a software license to the product element. Licenses cannot be set manually if the `Licensing.licensingMethod` config parameter is set to `LOCALSERVER` or `CTRLX`. Attempting to call this method in this case will result in a `LICENSING_METHOD_NOT_AVAILABLE` error.

- 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

7.4.4 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

7.4.5 licensingFeatureGetServedLicense

Requests a served license from the Revenera License Server. Once available the served license will be renewed automatically.

This method is only available if the `Licensing.licensingMethod` config parameter is set to `LOCALSERVER`. If this is not the case, attempting to call this method will result in a `LICENSING_METHOD_NOT_AVAILABLE` error. Note that switching the `Licensing.licensingMethod` to a value other than `LOCALSERVER` will automatically return the served license, if one is available.

If the config parameter `Licensing.getServedLicenseAutomatically` is true, then this API call will return a `LICENSING_LOCALSERVER_API_NOT_ALLOWED_IN_AUTOMATIC_MODE` error.

- QueryType : LicensingQueryMessage
- ResponseType : LicensingServedLicenseResponseMessage

Query Value

- sessionId: the session requesting the served license

Response Value

- response code as defined in the relevant section
- isInitialServedLicense: true if the requested served license is initial, false if a valid license has been existing but is renewed by the API.

7.4.6 licensingFeatureReturnServedLicense

Returns the served license to the Revenera License Server, making it available for other clients to use. The target ROKIT Locator itself will no longer be licensed.

This method is only available if the `Licensing.licensingMethod` config parameter is set to `LOCALSERVER`. If this is not the case, attempting to call this method will result in a `LICENSING_METHOD_NOT_AVAILABLE` error. Note that switching the `Licensing.licensingMethod` to a value other than `LOCALSERVER` will automatically return the served license, if one is available.

If the config parameter `Licensing.getServedLicenseAutomatically` is true, then this API call will return a `LICENSING_LOCALSERVER_API_NOT_ALLOWED_IN_AUTOMATIC_MODE` error.

- QueryType : LicensingQueryMessage
- ResponseType : LicensingResponseMessage

Query Value

- sessionId: the session requesting the served license return

Response Value

- response code as defined in the relevant section

7.4.7 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

LicensingInfoResponseMessage

- “responseCode”: ResponseCode
- “issueDate”: Timestamp
- “expiryDate” : Timestamp
- “vendorName” : AsciiCharacterString
- “hostId” : AsciiCharacterString
- “licenseVersion” : AsciiCharacterString
- “licenseStatus” : LicensingStatus
- “licenseType” : LicensingType
- “featureList” : array of LicensingFeature

LicensingServedLicenseResponseMessage

- “responseCode”: ResponseCode
- “isInitialServedLicense”: boolean

7.4.8 Objects

LicensingFeature

- “featureName”: AsciiCharacterString
- “featureVersion” : AsciiCharacterString
- “featureCapability” : LicensingFeatureCapability (deprecated)

7.4.9 Types

LicensingStatus (integer)

Label	Value	Comment
Valid	1	License is valid

Label	Value	Comment
Invalid	2	License is invalid. Possible reasons: hardware id mismatch, wrong license version, license is not imported after installation or factory reset, expired license
Not available	4	deprecated
Unknown	8	

LicensingType (integer)

License Model	Value
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	128

LicensingFeatureCapability (integer, deprecated) This type is deprecated. Its value should be ignored.

7.5 SYSTEM CONFIGURATION (MODULE: *CONFIG*)

If the minor version of the Config module has changed, additional parameters may have been added. To maintain backward compatibility, the user should therefore ignore parameters that are returned by the methods configList and configDefaultList but not listed in this API document.

7.5.1 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

7.5.2 configDefaultList

Returns a list of all configurable parameters, as well as their default values. The default values cannot be changed and are read-only.

- QueryType: ConfigQueryMessage
- ResponseType: ConfigEntryArrayResponseMessage

Query Value

- sessionId: the session requesting the config

Response Value

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

7.5.3 configSchemaGet

Returns the JSON Schema for any ConfigEntry that is part of ConfigEntryArrayQueryMessage and ConfigEntryArrayResponseMessage. The response contains the JSON Schema definitions of all configurable parameters and common JSON objects and types, that, for example, can be used to validate the parameters and values for the query method configSet.

- QueryType: ConfigQueryMessage
- ResponseType: ConfigSchemaResponseMessage

Query Value

- sessionId: the session requesting the config

Response Value

- response code as defined in the relevant section
- JSON schema for any ConfigEntry

7.5.4 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. This may take up to 20 seconds if the ROKIT Locator Client is already busy, such as during localization auto-start.

The configuration cannot be set while the ROKIT Locator Client is in a client control mode that uses the parameters. Specifically, the method will not change the configuration if the `LOC`, `REC`, `VISUALRECORDING`, `LASEROUTPUT`, or `MAP` control modes are in the RUN state. In this case, the method will fail with a `NOT_IN_REQUIRED_STATE` error. Before calling this method, these modes must therefore be returned to the READY state by calling the appropriate methods.

- 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

7.5.5 Messages

ConfigEntryArrayMessage

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

ConfigResponseMessage

- “responseCode”: ResponseCode

ConfigQueryMessage

- “sessionId”: SessionId

ConfigEntryArrayResponseMessage

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

ConfigSchemaResponseMessage

- “responseCode”: ResponseCode
- “configSchema”: string of JSON Schema

7.5.6 Objects

ConfigEntry

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

7.5.7 Types

See chapter 12 for a complete list of all available ConfigEntry types.

7.6 SYSTEM INFORMATION (MODULE: *ABOUTBUILD*)

7.6.1 aboutBuildList

Returns human-readable version information.

- QueryType: AboutBuildQueryMessage
- ResponseType: AboutBuildListResponseMessage

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 the ROKIT Locator.

7.6.2 aboutBuildVersion

Returns machine-readable version information of the ROKIT Locator component.

- QueryType: AboutBuildVersionQueryMessage
- ResponseType: AboutBuildVersionResponseMessage

Query Value None (empty JSON object)

Response Value

- responseCode: Response code as defined in the relevant section
- componentType: Type of the ROKIT Locator component sending this message.
- majorVersion: The component's major build version.
- minorVersion: The component's minor build version.
- patchVersion: The component's patch build version.

7.6.3 Messages

AboutBuildQueryMessage

- "sessionId": SessionId

AboutBuildListResponseMessage

- "responseCode": ResponseCode
- "aboutString": string

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

AboutBuildVersionResponseMessage

- "responseCode": ResponseCode
- "componentType": Type of the component (see AboutBuildComponentType)
- "majorVersion": The component's major version number
- "minorVersion": The component's minor version number
- "patchVersion": The component's patch version number

7.6.4 Types

AboutBuildComponentType (integer)

Label	Value	Description
UNDEFINED	0	Undefined component, should not occur
LOCATOR_CLIENT	1	The component ROKIT Locator Client
LOCATOR_SERVER	2	The component ROKIT Locator Server
LOCATOR_CLIENT_SUPPORT	3	The component ROKIT Locator Client Support
LOCATOR_SERVER_SUPPORT	4	The component ROKIT Locator Server Support

7.7 CERTIFICATE MANAGEMENT (MODULE: *CERTIFICATE*)

The user must read the Security chapter of the ROKIT Locator 1.10.5 first. All user certificates must fullfil the requirements listed there.

7.7.1 certificateSet

Sets the certificates for the communication between the components of the ROKIT Locator and the software implemented by the customer. The new certificates are used only after restarting the product element in question.

If the parameter `Certificate.enableRevocationList` is set to `false`, the revocation list will not be checked for validity and will not be used when establishing encrypted connections. In this case, the revocation list may even be empty. Note that subsequently setting `Certificate.enableRevocationList` back to `true` without first setting a valid certificate revocation list will disable encrypted communications. Consequently, a valid revocation list must be set using this method before re-enabling `Certificate.enableRevocationList`.

- 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

7.7.2 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

7.8 SYSTEM CONTROL (MODULE: *SYSTEM*)

7.8.1 systemShutdown

The product element receiving this message is requested to shut down. Depending on the ROKIT Locator 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

7.8.2 Messages

SystemQueryMessage

- “sessionId”: SessionId

SystemResponseMessage

- “responseCode”: ResponseCode

7.9 USER ACCOUNT MANAGEMENT (MODULE: *USER*)

7.9.1 userAdd

Adds a new user account to the ROKIT Locator.

- QueryType: UserAddMessage
- ResponseType: UserResponseMessage

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

7.9.2 userDelete

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

- QueryType: UserDeleteMessage
- ResponseType: UserResponseMessage

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

7.9.3 userChangePassword

Changes a user's password stored in the ROKIT Locator.

- QueryType: UserPasswordChangeMessage
- ResponseType: UserResponseMessage

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

7.9.4 userChangeOwnPassword

Changes a user's own password stored in the ROKIT Locator. Unlike userChangePassword, 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: UserOwnPasswordChangeMessage
- ResponseType: UserResponseMessage

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

7.9.5 userList

Retrieves the list of all user accounts known to the ROKIT Locator.

- QueryType: UserQueryMessage
- ResponseType: UserListResponseMessage

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 UserInfo

7.9.6 userListGroup

Retrieves the list of all user groups known to the ROKIT Locator.

- QueryType: UserQueryMessage
- ResponseType: UserGroupListResponseMessage

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 ROKIT Locator

7.9.7 Messages

UserAddMessage

- sessionId: SessionId
- userName: Username
- userPassword: Password
- userGroups: array of UserGroup

UserDeleteMessage

- sessionId: SessionId
- userName: Username

UserPasswordChangeMessage

- sessionId: SessionId
- userName: Username
- newUserPassword: Password

UserOwnPasswordChangeMessage

- sessionId: SessionId
- newUserPassword: Password
- oldUserPassword: Password

UserResponseMessage

- responseCode: ResponseCode

UserQueryMessage

- sessionId: SessionId

UserListResponseMessage

- responseCode: ResponseCode
- userInfoList: array of UserInfo

UserGroupListResponseMessage

- responseCode: ResponseCode
- userGroups: array of AsciiCharacterString

7.9.8 Objects

UserInfo

- “userName”: Username
- “userGroups”: array of UserGroup

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.

7.9.9 Types

UserGroup An AsciiCharacterString containing one of the following values:

- `admin`
- `user`
- `observer`

7.10 INTERNAL SETTINGS (MODULE: *INTERNAL*)

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.

7.10.1 internalUserDataSet

Set the user specific configuration (internal use only)

- QueryType: InternalUserDataSettingsMessage
- ResponseType: InternalResponseMessage

Query Value

- sessionId: the session setting the configuration
- the user specific configuration

Response Value

- response code as defined in the relevant section

7.10.2 internalUserDataGet

Get the user specific configuration (internal use only)

- QueryType: InternalQueryMessage
- ResponseType: InternalUserDataSettingsResponseMessage

Query Value

- sessionId: the session retrieving the configuration

Response Value

- response code as defined in the relevant section
- the stored user specific settings, or an empty string if an error occurred

7.10.3 Messages

InternalQueryMessage

- “sessionId”: SessionId

InternalResponseMessage

- “responseCode”: ResponseCode

InternalUserDataSettingsMessage

- “sessionId”: SessionId
- “settings”: InternalUserDataSettings

InternalUserDataSettingsResponseMessage

- “responseCode”: ResponseCode
- “settings”: InternalUserDataSettings or EmptyString in case of error

7.10.4 Objects

InternalUserDataSettings A string with a length of between 1 and 1,000,000 characters.

7.11 DATA EXCHANGE (MODULE: *DATAEXCHANGE*)

The API methods provided by this module allow for the exchange of data between different Rexroth ROKIT Locator instances. Data is first exported into a data exchange archive using the relevant API methods from the specific modules. These data exchange archives can then be downloaded and uploaded using the functionality provided by this API module.

Once a data exchange archive has been uploaded into a different instance of the ROKIT Locator, it can be imported using the API methods from the specific module. This will restore the data stored in the archive, completing its transfer from one instance of the ROKIT Locator to another. This functionality can be used to create and restore backup copies of specific data from a ROKIT Locator instance.

Additionally, this module offers methods for listing and deleting data exchange archives stored in a ROKIT Locator.

7.11.1 dataExchangeGetDownloadPath

Returns the download path for a specific data exchange archive. The data exchange archive can then be downloaded from the HTTP server under the given path.

- QueryType: DataExchangeArchiveNameMessage
- ResponseType: DataExchangeArchivePathResponseMessage

Query Value

- sessionId: the session requesting the archive
- archiveName: the name of the archive being requested

Response Value

- responseCode: response code as defined in the relevant section
- archiveNameUrl: a path including an authorization token, or an empty string on error

The path is the resource from which the Data Exchange Archive can be downloaded. This path contains a secutrity token, which ensures that only users who have succesfully called this API method may download a given data exchange archive.

The archive may be downloaded using an HTTP/HTTPS `GET` request for the URL `<PROTOCOL>://<PRODUCT_DOMAIN>:<PRODUCT_HTTP_PORT><PATH>`. Here, `PATH` is the path returend by this API method. For example, the actual upload URL may be `http://myhost:8080/data-exchange/ServerMap-mymap-20240604T113854Z-459a8e1c.tar?token=1a39037e-5177-4f9c-a449-94ee68d4b97a`.

Note: When downloading an archive via HTTP/HTTPS, keep in mind the remarks on the HTTP/HTTPS server.

7.11.2 dataExchangeGetUploadPath

Returns the upload path for a specific data exchange archive. The data exchange archive can then be uploaded from the HTTP server under the given path.

- QueryType: DataExchangeArchiveNameMessage
- ResponseType: DataExchangeArchivePathResponseMessage

Query Value

- sessionId: the session intending to upload the archive
- archiveName: the name of the archive intended to be uploaded

Response Value

- responseCode: response code as defined in the relevant section
- archiveNameUrl: a path including an authorization token, or an empty string on error

The path is the resource to which the Data Exchange Archive can be uploaded. This path contains a secutrity token, which ensures that only users who have succesfully called this API method may upload a data exchange archive.

The archive may be uploaded using an HTTP/HTTPS `PUT` request for the URL `<PROTOCOL>://<PRODUCT_DOMAIN>:<PRODUCT_HTTP_PORT><PATH>`. Here, `PATH` is the path returend by this API method. For example, the actual upload URL may be `http://myhost:8080/data-exchange/ServerMap-mymap-20240604T113854Z-459a8e1c.tar?token=1a39037e-5177-4f9c-a449-94ee68d4b97a`.

Note: When uploading an archive via HTTP/HTTPS, keep in mind the remarks on the HTTP/HTTPS server.

7.11.3 dataExchangeList

Provides a list of all Data Exchange Archives stored in the Rexroth ROKIT Locator instance.

- QueryType: DataExchangeQueryMessage
- ResponseType: DataExchangeArchiveListResponseMessage

Query Value

- sessionId: the session intending to upload the archive

Response Value

- responseCode: response code as defined in the relevant section
- archive: a list of DataExchangeArchive

Each entry in the list represents one Data Exchange Archive stored in the ROKIT Locator instance. Besides the name and size of the archive, each entry also contains additional information about the content of the archive. An entry's additional information is only valid if the "valid" flag of the entry is `true`.

If the "valid" flag of an entry is `false`, the additional information, except for the archive name and size (if non-zero), should be ignored. This case can occur if a Data Exchange Archive has been modified or damaged and the additional information cannot be read. In this case, the archive is still included in the list so that it can be deleted using dataExchangeDelete.

Additional information includes the name and type of the entity stored within the archive, the product and version from which the archive was exported, as well as the time at which the archive was created.

7.11.4 dataExchangeDelete

Deletes a given Data Exchange Archive from the Rexroth ROKIT Locator instance.

- QueryType: DataExchangeArchiveNameMessage
- ResponseType: DataExchangeResponseMessage

Query Value

- sessionId: the session intending to upload the archive
- archiveName: the name of the archive that should be deleted

Response Value

- responseCode: response code as defined in the relevant section

7.11.5 Messages

DataExchangeQueryMessage

- “sessionId”: SessionId

DataExchangeResponseMessage

- “responseCode”: ResponseCode

DataExchangeArchiveNameMessage

- “sessionId”: SessionId
- “archiveName”: DataExchangeArchiveName

DataExchangeArchivePathResponseMessage

- “responseCode”: ResponseCode
- “archiveNameUrl”: DataExchangeArchiveNameUrl

DataExchangeArchiveListResponseMessage

- “responseCode”: ResponseCode
- “archives”: List of DataExchangeArchive

7.11.6 Objects

DataExchangeArchive

- “name”: DataExchangeArchiveName
- “valid”: boolean
- “entityName”: DataExchangeEntityName
- “entityType”: AsciiCharacterString
- “productType”: AsciiCharacterString
- “productVersion”: DataExchangeProductVersion
- “createdAt”: Timestamp
- “size”: Size (in bytes)

DataExchangeProductVersion

- “majorVersion”: NonnegativeInteger64
- “minorVersion”: NonnegativeInteger64
- “patchVersion”: NonnegativeInteger64

7.11.7 Types

DataExchangeArchiveNameUrl (AsciiCharacterString, length > 0) A non-empty ASCII string of up to 512 characters in length.

7.11.8 DataExchangeEntityName (AsciiCharacterString, length > 0)

Archived entities 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 entity names.

8 ROKIT Locator Client – RPC Methods

8.1 RECORDING (MODULE: *CLIENTRECORDING*)

8.1.1 clientRecordingStart

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

Note that calling this method while either the VISUALRECORDING or MAP client control modes are in the RUN state will result in a `NOT_IN_REQUIRED_STATE` error.

To start a new recording, the ROKIT Locator Client requires at least 500 MiB of available storage space. Otherwise, a `CLIENTRECORDING_NOT_ENOUGH_SPACE` error will result.

- 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

8.1.2 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

8.1.3 clientRecordingStartVisualRecording

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

Note that calling this method while either the LOC, REC, or MAP client control modes are in the RUN state will result in a `NOT_IN_REQUIRED_STATE` error.

To start a new visual recording, the ROKIT Locator Client requires at least 500 MiB of available storage space. Otherwise, a `CLIENTRECORDING_NOT_ENOUGH_SPACE` error will result.

- 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

8.1.4 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

8.1.5 clientRecordingList

Retrieves a list of all recordings stored by the ROKIT Locator Client.

- QueryType: ClientRecordingQueryMessage
- ResponseType: ClientRecordingListResponseMessage

Query Value

- sessionId

Response Value

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

8.1.6 clientRecordingRename

Renames a recording stored by the ROKIT Locator 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

8.1.7 clientRecordingDelete

Deletes a recording stored by the ROKIT Locator 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

8.1.8 clientRecordingSetCurrentPose

Provides the ROKIT Locator Client with the current pose of the vehicle while recording. This pose must be determined through some external means, such as by moving the vehicle to known reference position.

- QueryType: ClientRecordingPoseMessage
- ResponseType: ClientRecordingResponseMessage

Query Value

- sessionId
- the pose of the vehicle at the time this method is called

Response Value

- response code as defined in the relevant section

8.1.9 clientRecordingExport

Exports an existing Recording into a Data Exchange Archive. This archive can be used together with the Data Exchange module to transfer the recording to another, compatible ROKIT Locator Client. It can also be used to restore the Recording at a later point. This method does not modify the target Recording in any way.

- QueryType: ClientRecordingNameMessage
- ResponseType: ClientRecordingArchiveNameResponseMessage

Query Value

- sessionId: the session exporting the recording
- clientRecordingName: the name of the recording that should be exported

Response Value

- responseCode: response code as defined in the relevant section
- archiveName: the name of the archive in which the exported Recording was stored. May be empty in case of an error

8.1.10 clientRecordingImport

Imports a Recording from a Data Exchange Archive. Suitable archives can be created by calling clientRecordingExport. They can also be transferred from a compatible ROKIT Locator Client by using the Data Exchange module. This method does not modify the target archive in any way.

If successful, the Recording stored within the archive is loaded into the ROKIT Locator Client and can immediately be used just like any other Recording. The name of the recording imported in this manner is returned as part of the response.

If a Recording of the same name already exists on the client, an `ENTITY_ALREADY_EXISTS` error will result. In this case, the response contains the name of the existing Recording that is preventing the archive from being imported. To import the archive, rename or delete the existing recording.

Note that Recordings may only be imported from archives that were created by compatible versions of the ROKIT Locator Client. Attempting to import a Recording from an archive that was created by an incompatible version of the ROKIT Locator Client will result in a `CLIENTRECORDING_ARCHIVE_INCOMPATIBLE` error. In general, archives are considered incompatible if they were created by a client with a newer (higher) version than the version of the client currently being used. In this case, the ROKIT Locator Client currently being used will have to be upgraded to a compatible version before the archive can be imported.

Attempting to import a Recording from an archive that contains another kind of entity (such as a Server Map) will result in a `CLIENTRECORDING_ARCHIVE_WRONG_TYPE` error.

Archive files are design to detect inadvertent changes or data corruption. If the archive has been changed since it was exported, a `CLIENTRECORDING_ARCHIVE_INVALID` may result. This ensures that a Recording that is imported from an archive has not been damaged or modified after it was exported. If the archive file has been severely damaged and cannot be read at all, a `FILE_ACCESS_FAILED` error may be returned instead.

Note that the ROKIT Locator Client itself never modifies the contents of an archive after it has been written. Therefore this kind of error is likely the result of problems in data storage or transmission that lie outside of the ROKIT Locator Client.

- QueryType: ClientRecordingArchiveNameMessage
- ResponseType: ClientRecordingNameResponseMessage

Query Value

- sessionId: the session importing the recording
- archiveName: the name of the archive from which the recording should be imported

Response Value

- responseCode: response code as defined in the relevant section
- clientRecordingName: the name of the recording that was imported from the archive. May be empty in case of an error

8.1.11 Messages

ClientRecordingQueryMessage

- “sessionId”: SessionId

ClientRecordingNameMessage

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

ClientRecordingNameResponseMessage

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

ClientRecordingRenameMessage

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

ClientRecordingPoseMessage

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

ClientRecordingListResponseMessage

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

ClientRecordingResponseMessage

- “responseCode”: ResponseCode

ClientRecordingArchiveNameMessage

- “sessionId”: SessionId
- “archiveName”: DataExchangeArchiveName

ClientRecordingArchiveNameResponseMessage

- “responseCode”: ResponseCode
- “archiveName”: DataExchangeArchiveName or EmptyString

8.2 CLIENT-SIDE MAP MANAGEMENT (MODULE: *CLIENTMAP*)

8.2.1 clientMapStart

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

Note that calling this method while either the VISUALRECORDING or REC client control modes are in the RUN state will result in a `NOT_IN_REQUIRED_STATE` error.

- 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

8.2.2 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

8.2.3 clientMapList

Retrieves all created maps stored by the ROKIT Locator 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

8.2.4 clientMapRename

Renames a map stored by the ROKIT Locator 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

8.2.5 clientMapDelete

Deletes a map stored by the ROKIT Locator 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

8.2.6 clientMapSend

Requests the ROKIT Locator Client to send the map to the ROKIT Locator 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 ROKIT Locator Server

Response Value

- response code as defined in the relevant section

8.2.7 clientMapGetInfo

Provides information about maps constructed by the ROKIT Locator Client.

- QueryType: ClientMapNameMessage

Query Value

- sessionId: the session requesting the map info
- name of the client-side map to get information about

Response Value

- responseCode: ResponseCode
- warnings: An array of AsciiCharacterString, containing warnings issued during map creation or modification.

8.2.8 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

8.3 CLIENT LOCALIZATION (MODULE: *CLIENTLOCALIZATION*)

8.3.1 clientLocalizationStart

Requests to start self-localizing within the map.

Note that calling this method while the VISUALRECORDING client control modes is in the RUN state will result in a `NOT_IN_REQUIRED_STATE` error.

- QueryType: ClientLocalizationQueryMessage
- ResponseType: ClientLocalizationResponseMessage

Query Value

- sessionId: the session requesting the mode change

Response Value

- response code as defined in the relevant section

8.3.2 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

8.3.3 clientLocalizationReset

Requests to reset the localization completely, making a stateless restart *without* reading new configurations and, thus, *without* changing the active map.

- QueryType: ClientLocalizationQueryMessage
- ResponseType: ClientLocalizationResponseMessage

Query Value

- sessionId: the session requesting the reset

Response Value

- response code as defined in the relevant section

8.3.4 clientLocalizationSetSeed

Provides the ROKIT Locator Client with an externally-derived estimate of the current vehicle pose. This estimate can assist the ROKIT Locator Client in self-localizing if the current pose is unknown. The current pose may be unknown if the ROKIT Locator Client has just started up or if the localization was lost. Seeds may only be set while the ROKIT Locator Client is self-localizing and the `LOC` control mode is in the `RUN` state. Otherwise, a `NOT_IN_REQUIRED_STATE` error will result. If a seed is set while the ROKIT Locator is temporarily not receiving any laser data, this seed will be used once laser data is once again received by the ROKIT Locator Client.

- QueryType: ClientLocalizationSeedMessage
- ResponseType: ClientLocalizationResponseMessage

Query Value

- sessionId: the session providing the pose estimate
- enforceSeed: When enforcing a seed, the ROKIT Locator Client will immediately attempt to self-localize at the given seed pose. If this is succesful, the localization status will switch to `LOCALIZED`. If the laser scans do not match the map at the given seed pose, the localization status will be set to `LOCALIZED_ODO_ONLY` instead. In either case, the localization pose generated by the ROKIT Locator Client will be set to the given seed pose.
  Additionally, the ROKIT Locator Client will discard all internally-generated estimates of the pose. Note that the ROKIT Locator Client may, however, generate additional pose estimates afterwards.
- uncertainSeed (optional, default value: false) : If set to true, the ROKIT Locator Client will consider the seed pose to be uncertain. In this case, the ROKIT Locator Client may deviate from the given pose by some amount while attempting to self-localize.
- seedPose: the pose that the ROKIT Locator Client should use to aid in localization

Response Value

- response code as defined in the relevant section

8.3.5 clientLocalizationDockingModeEnable

Enables the docking mode during localization. In this mode, the localization is optimized for slow and deliberate driving in close proximity to a docking station, charging station, or similar infrastructure. This mode should only be used in these situations. It should not be enabled while the vehicle is performing general movements through the environment.

Stopping or resetting the localization, for example by calling clientLocalizationStop or client-LocalizationReset, will disable the docking mode. This ensures that the ROKIT Locator Client always enters the localization in a well-defined state.

Note that the docking mode is always treated as enabled while the vehicle is localized within a `DOCKING` map zone.

If the docking mode is enabled, this will be indicated through the localization info flags.

The docking mode can only be changed while the localization is running. Calling this method while not in the `LOC` Client Control Mode) will result in a `NOT_IN_REQUIRED_STATE` error.

- QueryType: ClientLocalizationQueryMessage
- ResponseType: ClientLocalizationResponseMessage

Query Value

- sessionId: the session requesting the docking mode change

Response Value

- response code as defined in the relevant section

8.3.6 clientLocalizationDockingModeDisable

Disables the docking mode during localization.

Note that the docking mode is always treated as enabled while the vehicle is localized within a `DOCKING` map zone. If the vehicle is localized within such a zone, the docking mode will therefore remain active after this API method has been called until the vehicle leaves the `DOCKING` map zone.

The docking mode can only be changed while the localization is running. Calling this method while not in the `LOC` Client Control Mode) will result in a `NOT_IN_REQUIRED_STATE` error.

- QueryType: ClientLocalizationQueryMessage
- ResponseType: ClientLocalizationResponseMessage

Query Value

- sessionId: the session requesting the docking mode change

Response Value

- response code as defined in the relevant section

8.3.7 Messages

ClientLocalizationQueryMessage

- “sessionId”: SessionId

ClientLocalizationResponseMessage

- “responseCode”: ResponseCode

ClientLocalizationSeedMessage

- “sessionId”: SessionId
- “enforceSeed”: boolean
- “uncertainSeed”: boolean (optional)
- “seedPose”: Pose2D

8.4 MANUAL MAP ALIGNMENT (MODULE: *CLIENTMANUALALIGN*)

8.4.1 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

8.4.2 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

8.4.3 clientManualAlignGetMap

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

- QueryType: ClientManualAlignMapLevelMessage
- ResponseType: ClientManualAlignMapResponseMessage

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
- array of reflector markers in the prior map

8.4.4 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.

Manual map alignment mode needs to be started using clientManualAlignStart beforehand. This call will not automatically stop manual map aligment mode. Use clientManualAlignStop to do so if required.

- 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

8.4.5 Messages

ClientManualAlignQueryMessage

- “sessionId”: SessionId

ClientManualAlignResponseMessage

- “responseCode”: ResponseCode

ClientManualAlignMapNameResponseMessage

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

ClientManualAlignMapLevelMessage

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

ClientManualAlignMapResponseMessage

- “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
- “reflectors”: Array of PointCluster. Each cluster defines points within the data encoded in pointCloud that were classified as a single reflector marker. Its prototype describes the estimated position and normal of the reflector.

ClientManualAlignTransformMessage

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

ClientManualAlignOriginTransform2D

- “x”: IEEE754Double within the range of $[-10^4,10^4]$ (in meters)
- “y”: IEEE754Double within the range of $[-10^4,10^4]$ (in meters)
- “a”: IEEE754Double (in radians)

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

8.5 GLOBAL ALIGNMENT (MODULE: *CLIENTGLOBALALIGN*)

8.5.1 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

8.5.2 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 ROKIT Locator 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

8.5.3 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 ROKIT Locator Client is in the REC or VISUALRECORDING control modes.

It may be desirable to temporarily disable a landmark observation instead of permanently 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

8.5.4 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

8.5.5 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

8.5.6 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 VEHICLEstaticCalibSENSOR (see below).
- VEHICLEstaticCalibSENSOR: ClientGlobalAlignSensorTransform2D - Transformation between the coordinate system of the vehicle 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.
- “timestamp”(optional, default: the current time at which the ROKIT Locator Client received the observation): Timestamp - precise time at which the ClientGlobalAlignObservation was observed.

ClientGlobalAlignIdentifiedObservation

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

ClientGlobalAlignSensorTransform2D

- “x”: IEEE754Double within the range of $[-10^4, 10^4]$ (in meters)
- “y”: IEEE754Double within the range of $[-10^4, 10^4]$ (in meters)
- “a”: IEEE754Double (in radians)

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

ClientGlobalAlignLandmarkPose2D

- "x": IEEE754Double within the range of $[-10^4, 10^4]$ (in meters)
- "y": IEEE754Double within the range of $[-10^4, 10^4]$ (in meters)
- "a": IEEE754Double (in radians)

Describes the pose of a landmark within a 2D coordinate system, given by the position "x", "y" (in meters) and the orientation angle "a" (in radians).

8.5.7 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_OPTIMIZE_ORIENTATION	3	Calibration is optimized in orientation but fixed in translation

8.6 CLIENT-SIDE USER ACCOUNT MANAGEMENT (MODULE: *CLIENTUSER*)

The ClientUser module has been **deprecated** and will be removed in the next minor release version. Use the equivalent methods, messages, objects, and types from the User module instead. In the meantime, the existing methods, messages, objects, and types have been preserved for compatibility. However, these are only aliases for their counterparts from the User module.

8.6.1 clientUserAdd

Deprecated, alias for userAdd.

- QueryType: ClientUserAddMessage
- ResponseType: ClientUserResponseMessage

8.6.2 clientUserDelete

Deprecated, alias for userDelete.

- QueryType: ClientUserDeleteMessage
- ResponseType: ClientUserResponseMessage

8.6.3 clientUserChangePassword

Deprecated, alias for userChangePassword.

- QueryType: ClientUserPasswordChangeMessage
- ResponseType: ClientUserResponseMessage

8.6.4 clientUserChangeOwnPassword

Deprecated, alias for userChangeOwnPassword.

- QueryType: ClientUserOwnPasswordChangeMessage
- ResponseType: ClientUserResponseMessage

8.6.5 clientUserList

Deprecated, alias for userList.

- QueryType: ClientUserQueryMessage
- ResponseType: ClientUserListResponseMessage

8.6.6 clientUserListGroup

Deprecated, alias for userListGroup.

- QueryType: ClientUserQueryMessage
- ResponseType: ClientUserGroupListResponseMessage

8.6.7 Messages

ClientUserAddMessage

Deprecated, alias for UserAddMessage.

ClientUserDeleteMessage

Deprecated, alias for UserDeleteMessage.

ClientUserPasswordChangeMessage

Deprecated, alias for UserPasswordChangeMessage.

ClientUserOwnPasswordChangeMessage

Deprecated, alias for UserOwnPasswordChangeMessage.

ClientUserResponseMessage

Deprecated, alias for UserResponseMessage.

ClientUserQueryMessage

Deprecated, alias for UserQueryMessage.

ClientUserListResponseMessage

Deprecated, alias for UserListResponseMessage.

ClientUserGroupListResponseMessage

Deprecated, alias for UserGroupListResponseMessage.

8.6.8 Objects

ClientUserInfo

Deprecated, alias for UserInfo.

8.6.9 Types

ClientUserGroup

Deprecated, alias for UserGroup.

8.7 MAP EXPANSION (MODULE: *CLIENTEXPANDMAP*)

See the Mapexpansion Workflow for a detailed description of the required steps.

8.7.1 clientExpandMapEnable

Enables map expansion and requests to enter the control mode EXPANDMAP. The feature Map Expansion shows effect when the ROKIT Locator Client is in the REC, VISUALRECORDING or MAP control mode:

If map expansion is enabled and a valid prior map is set, client recordings created in the REC and VISUALRECORDING modes will be map expansion recordings. These recordings will contain the necessary information to expand the prior map.

Regardless of whether map expansion is enabled or not, building a map from a map expansion recording will result in a map that expands the prior map with the information from the recording. This expansion uses the prior map which the ROKIT Locator Client used at the time the recording was made.

A recording that is not a map expansion recording cannot be used to expand an exising map, as this recording contains no information about the prior map. Instead, a regular non-expanded map will be built from the recording.

- QueryType: ClientExpandMapEnableMessage
- ResponseType: ClientExpandMapResponseMessage

Query Value

- “sessionId”: SessionId
- “priorMapName”: ClientExpandMapPriorMapName. The map with the “priorMapName” has to be loaded in the ROKIT Locator Client, e.g. the localization has been started at least once with this map with a valid pose (localization status = LOCALIZED) in the ClientLocalizationPoseInterface.

Response Value

- response code as defined in the relevant section

8.7.2 clientExpandMapDisable

Disables map expansion and requests to leave the control mode EXPANDMAP.

- QueryType: ClientExpandMapQueryMessage
- ResponseType: ClientExpandMapResponseMessage

Query Value

- sessionId: SessionId

Response Value

- response code as defined in the relevant section

8.7.3 clientExpandMapAddOverwriteZone

Adds a zone in which the prior map will be overwritten by the recording. The expansion is limited to the marked area. This API can only be called while recording (VISUALRECORDING and REC in Client Control Mode).
Note: Multiple identical zones with the same name and polygon can be added. They can be modified after the recording is finished using clientExpandMapReplaceOverwriteZones or clientExpandMapDeleteOverwriteZones.

- QueryType: ClientExpandMapOverwriteZoneMessage
- ResponseType: ClientExpandMapOverwriteZoneIdResponseMessage

Query Value

- sessionId: SessionId
- zone: The zone to add to the current recording

Response Value

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

8.7.4 clientExpandMapDeleteOverwriteZones

Deletes overwrite areas in a saved recording. Note that the original recording is not modified: Instead, the method creates a copy of the recording where the specified overwrite areas have been deleted. The name of this newly modified recording is returned as part of the response value. This method may not be called while the ROKIT Locator Client is in the REC or VISUALRECORDING control modes.

- QueryType: ClientExpandMapOverwriteZoneIdsMessage
- ResponseType: ClientExpandMapRecordingNameResponseMessage

Query Value

- sessionId: SessionId
- the name of the recording from which to delete the existing zones
- the unique IDs of the specific zones 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 zones have been deleted

8.7.5 clientExpandMapReplaceOverwriteZones

Replaces overwrite areas in a saved recording. Note that the original recording is not modified: Instead, the method creates a copy of the recording where the specified overwrite areas 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 ROKIT Locator Client is in the REC or VISUALRECORDING control modes.

- QueryType: ClientExpandMapReplaceOverwriteZonesMessage
- ResponseType: ClientExpandMapRecordingNameResponseMessage

Query Value

- sessionId: SessionId
- the name of the recording within which to replace the existing zones
- array containing the unique IDs of the specific zones that should be replaced, as well as the new zones 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 specified zones have been replaced

8.7.6 clientExpandMapListOverwriteZones

Lists overwrite areas in an existing and finalized recording.

- QueryType: ClientExpandMapRecordingNameMessage
- ResponseType: ClientExpandMapIdentifiedOverwriteZoneArrayResponseMessage

Query Value

- sessionId: SessionId
- the name of the recording for which to retrieve the zones

Response Value

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

8.7.7 clientExpandMapGetPriorMap

Gets the prior map in an existing and finalized recording.

- QueryType: ClientExpandMapRecordingNameMessage
- ResponseType: ClientExpandMapPriorMapResponseMessage

Query Value

- sessionId: SessionId
- the name of the recording for which to retrieve the prior map

Response Value

- response code as defined in the relevant section
- name of the prior map
- point cloud of the prior map
- size of the point cloud
- array of reflector markers in the prior map

8.7.8 Messages

ClientExpandMapResponseMessage

- responseCode: ResponseCode

ClientExpandMapQueryMessage

- sessionId: SessionId

ClientExpandMapEnableMessage

- sessionId: SessionId
- priorMapName: ClientExpandMapPriorMapName

ClientExpandMapOverwriteZoneMessage

- sessionId: SessionId
- zone: ClientExpandMapOverwriteZone

ClientExpandMapOverwriteZoneIdResponseMessage

- responseCode: ResponseCode
- zoneId: ClientExpandMapOverwriteZoneId

ClientExpandMapOverwriteZoneIdsMessage

- sessionId: SessionId
- recordingName: RecordingName
- zoneIds: a non empty array of unique ClientExpandMapOverwriteZoneId

ClientExpandMapRecordingNameResponseMessage

- responseCode: ResponseCode
- recordingName: RecordingName or EmptyString

ClientExpandMapReplaceOverwriteZonesMessage

- sessionId: SessionId
- recordingName: RecordingName
- replacementZones: array of ClientExpandMapIdentifiedOverwriteZone

ClientExpandMapRecordingNameMessage

- sessionId: SessionId
- recordingName: RecordingName

ClientExpandMapIdentifiedOverwriteZoneArrayResponseMessage

- responseCode: ResponseCode
- zones: array of ClientExpandMapIdentifiedOverwriteZone

ClientExpandMapPriorMapResponseMessage

- responseCode: ResponseCode
- priorMapName: ClientExpandMapPriorMapName
- pointCloud: Container of media type application/PointCloud2D.
- pointCloudSize: Size
- reflectors: Array of PointCluster. Each cluster defines points within the data encoded in pointCloud that were classified as a single reflector marker. Its prototype describes the estimated position and normal of the reflector.

8.7.9 Objects

ClientExpandMapOverwriteZoneName(AsciiCharacterString) User chosen name of the zone consisting of 1 to 240 characters.

ClientExpandMapOverwriteZone

- zoneName: ClientExpandMapOverwriteZoneName
- zoneType: ClientExpandMapOverwriteZoneType
- polygonPoints: array of Point2D

ClientExpandMapIdentifiedOverwriteZone

- zoneId: ClientExpandMapOverwriteZoneId
- zone: ClientExpandMapOverwriteZone

8.7.10 Types

ClientExpandMapPriorMapName (AsciiCharacterString, length > 0)

ClientExpandMapOverwriteZoneId (Integer64) Zone IDs encoded as integers.

ClientExpandMapOverwriteZoneType (integer)

Label	Value	Description
ENFORCE	1	Zone is used in map expansion to overwrite the specified area in the prior map.
IGNORE	2	All zones of this type are ignored

8.8 SENSOR (MODULE: *CLIENTSENSOR*)

8.8.1 clientSensorGetLaserScan

This method is only available while the ROKIT Locator Client is in the LASEROUTPUT client control mode. If this is not the case, calling this method will result in a `NOT_IN_REQUIRED_STATE` error.

Retrieves the next available scan from the specified sensor. Attempting to retrieve a scan from the second laser if the second laser is not enabled will result in a `SENSOR_NOT_AVAILABLE` error.

- QueryType: ClientSensorLaserIndexMessage
- ResponseType: ClientSensorLaserScanResponseMessage

Query Value

- sessionId
- laserIndex (optional, default value: 0) : Specifies the laser for which the scan should be retrieved. A value of 0 retrieves a scan from the first laser, while a value of 1 retrieves a scan from the second laser. Values of 2-5 retrieve scans from the auxiliary lasers numbered 3-6, respectively.

Response Value

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

8.8.2 clientSensorLaserOutputStart

Requests to enter the client control mode LASEROUTPUT.

- QueryType: ClientSensorQueryMessage
- ResponseType: ClientSensorResponseMessage

Query Value

- sessionId: the session requesting the mode change

Response Value

- response code as defined in the relevant section

8.8.3 clientSensorLaserOutputStop

Requests to leave the client control mode LASEROUTPUT.

- QueryType: ClientSensorQueryMessage
- ResponseType: ClientSensorResponseMessage

Query Value

- sessionId: the session requesting the mode change

Response Value

- response code as defined in the relevant section

8.8.4 clientSensorCalibrateDualLaser

Uses sensor data from a recording to automatically calculate and return the calibration parameters for the second laser.

This method may not be called while either the `LOC`, `REC`, or `VISUALRECORDING` control modes are in `RUN` state. Doing so will result in a `NOT_IN_REQUIRED_STATE` error.

The specified recording must contain sufficient sensor data for both laser sensors. Without this data, the calibration will fail with a `CLIENTSENSOR_CALIBRATION_FAILED` error. It will fail with `CLIENTSENSOR_CALIBRATION_NO_VALID_LASER2_DATA` in case no data originating from the secondary laser is contained in the data or the data is discarded due to quality or synchronization problems.

Note: This method will only return the correct calibration for the second laser if the calibration for the first laser has already been to the correct value.

Note: This method returns 3D calibration parameters with six degrees of freedom (6 DOF). However, as both lasers are purely 2D sensors, the calibration result will be 2D only. The parameters corresponding to the additional degrees of freedom (z, roll, pitch) will thus always be zero.

- QueryType: ClientSensorCalibrationMessage
- ResponseType: ClientSensorCalibrationResponseMessage

Query Value

- sessionId: the session requesting the calibration
- name of the recording containing the sensor data used for calibration

Response Value

- response code as defined in the relevant section
- Calibration parameters for the second laser. Note that angles are given in degrees, just like the calibration parameters stored within the system configuration.

8.8.5 clientSensorCalibrateOdometry

Uses sensor data from a recording to automatically calculate and return the calibration parameters for the external odometry.

This method may not be called while either the `LOC`, `REC`, or `VISUALRECORDING` control modes are in `RUN` state. Doing so will result in a `NOT_IN_REQUIRED_STATE` error.

The specified recording must contain sufficient sensor data for both the laser and the external odometry. Without this data, the calibration will fail with a `CLIENTSENSOR_CALIBRATION_FAILED` error. It will fail with `CLIENTSENSOR_CALIBRATION_NO_VALID_ODOMETRY_DATA` in case no data originating from the external odometry sensor is contained within the data or it is discarded due to synchronization and quality issues.

Note: This method will only return the correct calibration for the odometry if the calibration(s) for the laser sensors already been to the correct value.

Note: This method returns 3D calibration parameters with six degrees of freedom (6 DOF). However, as both the laser and odometry are purely 2D sensors, the calibration result will be 2D only. The parameters corresponding to the additional degrees of freedom (z, roll, pitch) will thus always be zero.

- QueryType: ClientSensorCalibrationMessage
- ResponseType: ClientSensorCalibrationResponseMessage

Query Value

- sessionId: the session requesting the calibration
- name of the recording containing the sensor data used for calibration

Response Value

- response code as defined in the relevant section
- Calibration parameters for the external odometry. Note that angles are given in degrees, just like the calibration parameters stored within the system configuration.

8.8.6 Messages

ClientSensorQueryMessage

- “sessionId”: SessionId

ClientSensorResponseMessage

- “responseCode”: ResponseCode

ClientSensorLaserIndexMessage

- “sessionId”: SessionId
- “laserIndex”: integer, either 0, 1, 2, 3, 4, or 5 (optional)

ClientSensorLaserScanResponseMessage This message contains the data of a single laser scan from a rotating laser scanner. In general, the content of this message is analogous to that of a ClientSensorLaserDatagram.

If the scanner that captured the scan is mounted upside-down and the configuration parameter `mirrorLaserScans` is enabled for the scanner in question, then the scan data within the message is already corrected for the fact that the laser scanner is inverted. When creating a point cloud from the measurements within this message, the fact that the scanner is upside down thus does not have to be considered.

- “responseCode”: ResponseCode
- “ranges”: array of IEEE754Double (in meters)
- “intensities”: array of IEEE754Double
- “numBeams”: Size
- “angleInc”: IEEE754Double (in radians)
- “minIntensity”: IEEE754Double
- “maxIntensity”: IEEE754Double
- “timeStart”: Timestamp
- “durationBeam”: TimeInterval
- “durationScan”: TimeInterval
- “durationRotation”: TimeInterval
- “scanNum”: Size
- “hasIntensities”: boolean
- “angleStart”: IEEE754Double (in radians)
- “angleEnd”: IEEE754Double (in radians)
- “minRange”: IEEE754Double (in meters)
- “maxRange”: IEEE754Double (in meters)

ClientSensorCalibrationMessage

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

ClientSensorCalibrationResponseMessage

- “responseCode”: ResponseCode
- “x”: IEEE754Double (in meters)
- “y”: IEEE754Double (in meters)
- “z”: IEEE754Double (in meters)
- “roll”: IEEE754Double (in degrees)
- “pitch”: IEEE754Double (in degrees)
- “yaw”: IEEE754Double (in degrees)

9 ROKIT Locator Server – RPC Methods

The main task of the ROKIT Locator Server is to store, update, and distribute the maps used by the ROKIT Locator 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 layer is uniquely 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 ROKIT Locator Server. Such maps have to be built using the ROKIT Locator Client’s map building process.

9.1 SERVER-SIDE MAP MANAGEMENT (MODULE: *SERVERMAP*)

9.1.1 serverMapGetInfo

Returns all information about a given map

- QueryType: ServerMapNameMessage
- ResponseType: ServerMapInfoResponseVersionedMessage

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 ROKIT Locator Server’s memory
- type: currently not used
- fingerprint: A unique fingerprint calculated from the map creation, update, and receive time. Used to identify a specific state of the map in time.
- “sentAt”: the time the map was last sent by the ROKIT Locator Client to the ROKIT Locator Server
- “receivedAt”: the time the map was originally received by the ROKIT Locator Server from the ROKIT Locator 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
- “initialMapLaserTypes”: array of laser type used in creating the initial map, see additional laser type info.
- “currentMapLaserTypes”: array of laser type used in creating the current map, see additional laser type info.

- “warnings”: array of strings containing warnings issued during map creation or modification.

The “initialMapLaserTypes” can be used to determine which laser was used to create the map. This helps to avoid using a different laser scanner on a map, than the laser scanner used to create the map. The “currentMapLaserTypes” can be used to ensure that a map only contains laser scan information from one type of laser scanner. It can be used to ensure that only one type of laser scanner was used to create the map and that only the same type of laser scanner contributed (e.g. via map updates) to the map. Different laser scanners on the same map are not supported and may lead to false localization and map corruption issues.

9.1.2 serverMapList

Requests a list of all maps managed by the ROKIT Locator 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 ROKIT Locator Server

9.1.3 serverMapRename

Renames a map managed by the ROKIT Locator 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

9.1.4 serverMapDelete

Deletes a map managed by the ROKIT Locator 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

9.1.5 serverMapGetMap

Retrieves a point cloud representation of a map managed by the ROKIT Locator Server.

- QueryType: ServerMapNameMessage
- ResponseType: ServerMapMapResponseMessage

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
- array of reflector markers found in the map

9.1.6 serverMapGetImage

Retrieves a raster image of the map with a given image size. Note that the aspect ratio of the map usually does not match the ratio of the requested image size. In this case, the image is scaled so that either the width or height matches the requested size, while the image size along the other axis remains at or below its requested size. In other words, the resulting image size is chosen to be as large as possible within the specified limits while also maintaining the aspect ratio of the map.

- QueryType: ServerMapImageSizeMessage
- ResponseType: ServerMapPngImageResponseMessage

Query Value

- sessionId: the session requesting the image
- width: integer in range from [1, 20000] (in pixels)
- height: integer in range from [1, 20000] (in pixels)
- 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 meters)
- height (in meters)
- container holding the requested image in PNG format

9.1.7 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 meters)
- height (in meters)
- container holding the requested image in PNG format

9.1.8 serverMapGetMapThumbnail

Retrieves a thumbnail-sized raster image of the map. Note that the pose of the image origin and map size within the response are not set and are always 0.

- 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 (not used, always 0)
- width (not used, always 0)
- height (not used, always 0)
- container holding the requested image in PNG format

9.1.9 serverMapAddZone

Adds a specified zone to the specified map.

- QueryType: ServerMapAddZoneMessage
- ResponseType: ServerMapAddZoneResponseMessage

Query Value

- sessionId: the session adding the zone to the specified map
- serverMapName: name of the map to which the zone shall be added
- mapZone: the zone that shall be added

Response Value

- response code as defined in the relevant section

9.1.10 serverMapDeleteZone

Deletes a specified zone contained in a specified map.

- QueryType: ServerMapDeleteZoneMessage
- ResponseType: ServerMapResponseMessage

Query Value

- sessionId: the session deleting the zone
- serverMapName: name of the map containing the zone that shall be deleted
- zoneId: unique identifier of the zone that shall be deleted

Response Value

- response code as defined in the relevant section

9.1.11 serverMapSetZoneName

Changes the name of a specified zone in a specified map.

- QueryType: ServerMapSetZoneNameMessage
- ResponseType: ServerMapResponseMessage

Query Value

- sessionId: the session changing the zone name
- serverMapName: name of the map containing the zone that shall be renamed
- zoneId: unique identifier of the zone that shall be renamed
- zoneName: new name of the zone

Response Value

- response code as defined in the relevant section

9.1.12 serverMapSetZoneType

Changes the type of a specified zone in a specified map.

- QueryType: ServerMapSetZoneTypeMessage
- ResponseType: ServerMapResponseMessage

Query Value

- sessionId: the session changing the zoneType
- serverMapName: name of the map containing the zone who’s type shall be changed
- zoneId: unique identifier of the zone who’s type shall be changed
- zoneType: new type of the zone

Response Value

- response code as defined in the relevant section

9.1.13 serverMapSetZonePolygon

Changes the shape of a specified zone in a specified map.

- QueryType: ServerMapSetZonePolygonMessage
- ResponseType: ServerMapResponseMessage

Query Value

- sessionId: the session changing the shape of the zone
- serverMapName: name of the map containing the zone who's shape shall be changed
- zoneId: unique identifier of the zone who's shape shall be changed
- polygonPoints: polygon defining the new shape of the zone

Response Value

- response code as defined in the relevant section

9.1.14 serverMapExport

Exports an existing Server Map into a Data Exchange Archive. This archive can be used together with the Data Exchange module to transfer the map to another, compatible ROKIT Locator Server. It can also be used to restore the Server Map at a later point. This method does not modify the target Server Map in any way.

- QueryType: ServerMapNameMessage
- ResponseType: ServerMapArchiveNameResponseMessage

Query Value

- sessionId: the session exporting the map
- serverMapName: the name of the map that should be exported

Response Value

- responseCode: response code as defined in the relevant section
- archiveName: the name of the archive in which the exported Server Map was stored. May be empty in case of an error

9.1.15 serverMapImport

Imports a Server Map from a Data Exchange Archive. Suitable archives can be created by calling serverMapExport. They can also be transferred from a compatible ROKIT Locator Server by using the Data Exchange module. This method does not modify the target archive in any way.

If successful, the Server Map stored within the archive is loaded into the ROKIT Locator Server and can immediately be used just like any other Server Map. The name of the map imported in this manner is returned as part of the response.

If a Server Map of the same name already exists on the server, an `ENTITY_ALREADY_EXISTS` error will result. In this case, the response contains the name of the existing Server Map that is preventing the archive from being imported. To import the archive, rename or delete the existing map.

Note that Server Maps may only be imported from archives that were created by compatible versions of the ROKIT Locator Server. Attempting to import a Server Map from an archive that was created by an incompatible version of the ROKIT Locator Server will result in a `SERVERMAP_ARCHIVE_INCOMPATIBLE` error. In general, archives are considered incompatible if they were created by a server with a newer (higher) version than the version of the server currently being used. In this case, the ROKIT Locator Server currently being used will have to be upgraded to a compatible version before the archive can be imported.

Attempting to import a Server Map from an archive that contains another kind of entity (such as a Recording) will result in a `SERVERMAP_ARCHIVE_WRONG_TYPE` error.

Archive files are design to detect inadvertent changes or data corruption. If the archive has been changed since it was exported, a `SERVERMAP_ARCHIVE_INVALID` may result. This ensures that a Server Map that is imported from an archive has not been damaged or modified after it was exported. If the archive file has been severely damaged and cannot be read at all, a `FILE_ACCESS_FAILED` error may be returned instead.

Note that the ROKIT Locator Server itself never modifies the contents of an archive after it has been written. Therefore this kind of error is likely the result of problems in data storage or transmission that lie outside of the ROKIT Locator Server.

- QueryType: ServerMapArchiveNameMessage
- ResponseType: ServerMapNameResponseMessage

Query Value

- sessionId: the session importing the map
- archiveName: the name of the archive from which the map should be imported

Response Value

- responseCode: response code as defined in the relevant section
- serverMapName: the name of the map that was imported from the archive. May be empty in case of an error

9.1.16 Messages

ServerMapQueryMessage

- sessionId: SessionId

ServerMapNameMessage

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

ServerMapNameResponseMessage

- “responseCode”: ResponseCode
- “serverMapName”: ServerMapName or EmptyString

ServerMapLevelMessage

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

ServerMapRenameMessage

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

ServerMapListResponseMessage

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

ServerMapInfoResponseVersionedMessage

- “responseCode”: ResponseCode
- “level”: Integer64
- “loaded”: boolean
- “type”: Integer64
- “fingerprint”: Integer64
- “createdAt”: Timestamp
- “updatedAt”: Timestamp
- “sentAt”: Timestamp
- “receivedAt”: Timestamp
- “initialMapLaserTypes”: LaserType
- “currentMapLaserTypes”: LaserType
- “warnings”: Array of AsciiCharacterString

ServerMapMapResponseMessage

- “responseCode”: ResponseCode
- “pointCloud”: Container of media type application/PointCloud2D.
- “pointCloudSize”: Size
- “reflectors”: Array of PointCluster. Each cluster defines points within the data encoded in pointCloud that were classified as a single reflector marker. Its prototype describes the estimated position and normal of the reflector.
- “zones”: Array of ServerMapIdentifiedZone of all zones contained in this map

ServerMapImageSizeMessage

- “sessionId”: SessionId
- “width”: Size, within the range of [1, 20000]
- “height”: Size, within the range of [1, 20000]
- “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 (in meters).
- “height”: IEEE754Double, greater than or equal to 0; height of the map in the Cartesian coordinates system (in meters).
- “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.

ServerMapResponseMessage

- “responseCode”: ResponseCode

ServerMapAddZoneResponseMessage

- “responseCode”: ResponseCode
- “zoneId”: Integer64; unique identifier of the zone within the map

ServerMapAddZoneMessage

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

ServerMapDeleteZoneMessage

- “sessionId”: SessionId
- “serverMapName”: ServerMapName; name of the map which contains the relevant zone
- “zoneId”: Integer64; unique identifier of a zone within the map

ServerMapSetZoneNameMessage

- “sessionId”: SessionId
- “serverMapName”: ServerMapName; name of the map which contains the relevant zone
- “zoneId”: Integer64; unique identifier of a zone within the map
- “zoneName”: ServerMapZoneName; desired new name of the zone with the given identifier

ServerMapSetZoneTypeMessage

- “sessionId”: SessionId
- “serverMapName”: ServerMapName; name of the map which contains the relevant zone
- “zoneId”: Integer64; unique identifier of a zone within the map
- “zoneType”: ServerMapZoneType; desired new type of the zone with the given identifier

ServerMapSetZonePolygonMessage

- “sessionId”: SessionId
- “serverMapName”: ServerMapName; name of the map which contains the relevant zone
- “zoneId”: Integer64; unique identifier of a zone within the map
- “polygonPoints”: array of Point2D to define the new shape of the zone

ServerMapArchiveNameMessage

- “sessionId”: SessionId
- “archiveName”: DataExchangeArchiveName

ServerMapArchiveNameResponseMessage

- “responseCode”: ResponseCode
- “archiveName”: DataExchangeArchiveName or EmptyString

9.1.17 Objects

ServerMapZone Represents a zone added to a specific map.

- zoneName: AsciiCharacterString, user chosen name of the zone
- zoneType: ServerMapZoneType, type of the zone
- polygonPoints: array of Point2D to define the shape of the zone

ServerMapIdentifiedZone A zone added to a specific map and its respective unique id.

- zoneId: Unique identifier of a zone.
- mapZone: ServerMapZone, represents a zone added to a specific map.

9.1.18 Types

LaserType (integer)

Label	Hex Value	Description
UNKNOWN	0x00000000	Unknown laser
SIMPLE	0x00000001	Simple laser interface
UNITY	0x00010000	Unity laser scanner
DUMMYS300	0x00020000	Dummy laser scanner for internal use
OMRON	0x00030000	OMRON laser scanner
IDEC	0x00040000	IDEC laser scanner
LEISHEN	0x00050000	Leishen laser scanner (deprecated)
SICK	0x00060000	SICK laser scanner
SICKLMS	0x00060001	SICK LMS laser scanner
SICKCOLA2	0x00060002	SICK laser scanner using SICK's CoLa2 interface
SICKCOLA2 UDP	0x00060003	SICK laser scanner using SICK's CoLa2 UDP interface
SICKCOLA2_ MICS3	0x00060004	SICK microScan3 laser scanner using SICK's CoLa2 interface
SICKCOLA2_ NANS3	0x00060005	SICK nanoScan3 laser scanner using SICK's CoLa2 interface
SICKCOLA2 UDP_MICS3	0x00060006	SICK microScan3 laser scanner using SICK's CoLa2 UDP interface
SICKCOLA2 UDP_NANS3	0x00060007	SICK nanoScan3 laser scanner using SICK's CoLa2 UDP interface
SICKPICOSCAN UDP	0x00060008	SICK picoScan laser scanner using SICK's picoScan UDP interface
PepperlFuchs	0x00070000	Pepperl+Fuchs laser scanner
PFR2000	0x00070001	Pepperl+Fuchs PFR2000 laser scanner
KEYENCE	0x00090000	Keyence laser scanner
KEYENCE_SZ_V_ UDP	0x00090001	Keyence SZ-V laser scanner using the UDP interface

ServerMapZoneName (AsciiCharacterString) User chosen name of the zone consisting of 1 to 240 characters

ServerMapZoneType (integer)

Label	Value	Description
IGNORE	2	All zones of this type are ignored.
STATIC	4	No points are added or removed from the map inside this zone.
RESTRICTED_ RELOCALIZATION	5	While localized inside this zone, the ROKIT Locator Client will not perform an automatic relocalization.
DYNAMIC	6	Reserved but unused.
DOCKING	7	While localized inside this zone, the ROKIT Locator Client will localize with docking mode enabled.

9.2 SERVER-SIDE POST-MAPPING SCAN ALIGNMENT (MODULE: *SERVERPOSTALIGN*)

The ServerPostAlign module allows users to manually edit a map. Each map consists of a number of laser scans which, when combined, result in the complete map. Using this module, selected laser scans can be aligned to a specified pose. Other laser scans in the map will automatically be aligned to be consistent with the specified poses.

9.2.1 serverPostAlignMap

Edits a map by moving a selected set of laser scans to the given poses. The response contains the laser scans that make up the map, including their poses and scan points. Scan points are given relative to the laser scanner. Calling this method without any new poses will return the existing map without any new alignments. Optionally, the resulting aligned map can be saved as a new map.

This method is only supported for server maps that were sent to the server using ROKIT Locator Client version 1.6 or newer. Attempting to use this method with a map that was sent to the server using an older version will result in a `SERVERPOSTALIGN_MAP_INCOMPATIBLE` error.

This method does not support maps that were created using map expansion. Attempting to use this method with such an expanded map will result in a `SERVERPOSTALIGN_MAP_INCOMPATIBLE` error.

This method overrides all information from Global Alignment as well as absolute pose data. The resulting map will not incorporate any information from Global Alignment or absolute pose data.

This method works only on new maps, directly after mapping. Any map updates that have taken place between sending the map to the server and applying post alignment will be discarded. There is no additional warning to indicate this loss of information.

Trying to align the same scan to two different poses at the same time will result in a `SERVERPOSTALIGN_DUPLICATE_SCAN_ID` error. Similarly, attempting to align a scan that does not exist in the map results in a `SERVERPOSTALIGN_INVALID_SCAN_ID` error.

- QueryType: ServerPostAlignMapMessage
- ResponseType: ServerPostAlignMapResponseMessage

Query Value

- sessionId: the session editing the map
- serverMapName: name of the map on which the method works. Note that this map is not actually changed by the operation.
- alignedServerMapName (optional): name under which the resulting map is stored. If a map with this name already exists, a `ENTITY_ALREADY_EXISTS` error is returned. If this argument is omitted, no map will be written.
- alignedScanPoses: Each entry in this array corresponds to one scan. Each scan with a given scan id will be aligned to the given pose. If the pose is `null`, the scan will revert to being unaligned. The scan ids correspond to those ids given in the response message.

Response Value

- responseCode: response code as defined in the relevant section
- scans: The scans within the map after the given alignment has been applied. Each entry within this array corresponds to a scan within the resulting map, and contains the scan's id, pose, alignment type, and laser scan points. Note that laser scan points are given relative to the scan pose, not relative to the map.

9.2.2 serverPostAlignMapCompressed

This method performs the same operation as serverPostAlignMap. When calling this method, the same restrictions apply as when calling serverPostAlignMap.

This method also accepts the same query value. However, the response of this method uses a compressed response message. While the values contained within the response are the same as for serverPostAlignMap, the compressed format is shorter and the response is faster.

- QueryType: ServerPostAlignMapMessage
- ResponseType: ServerPostAlignMapResponseCompressedMessage

Query Value Identical to the query value of serverPostAlignMap.

Response Value

- responseCode: response code as defined in the relevant section
- scansSize: The number of scans contained in the response
- scans: The scans within the map after the given alignment has been applied. This base64-encoded Container consists of an array with scansSize entries in a binary representation. Each entry within the array corresponds to a scan within the resulting map, and contains the scan's id, pose, alignment type, and laser scan points. Note that laser scan points are given relative to the scan pose, not relative to the map.

9.2.3 Messages

ServerPostAlignMapMessage

- “sessionId”: SessionId
- “serverMapName”: ServerMapName
- “alignedServerMapName”: ServerMapName (optional)
- “alignedScanPoses”: array of ServerPostAlignScanPose

ServerPostAlignMapResponseMessage

- “responseCode”: ResponseCode
- “scans”: array of ServerPostAlignScan

ServerPostAlignMapCompressedResponseMessage

- “responseCode”: ResponseCode
- “scansSize”: NonnegativeInteger64
- “scans”: Container with contentEncoding ”base64” and and contentMediaType application/AlignedScans.

9.2.4 Objects

ServerPostAlignPose2D

- “x”: IEEE754Double within the range of $[-10^4, 10^4]$ (in meters)
- “y”: IEEE754Double within the range of $[-10^4, 10^4]$ (in meters)
- “a”: IEEE754Double (in radians)

Describes the pose within a 2D coordinate system to which a scan should be aligned, given by the position “x”, “y” (in meters) and the orientation angle “a” (in radians).

ServerPostAlignScanPose

- “id”: NonnegativeInteger64
- “alignPose”: ServerPostAlignPose2D or `null`

Specifies that the scan with the given “id” should be aligned to the given “alignPose”.

If “alignPose” is `null`, specifies that the scan should not be aligned. This removes any previous alignment of this scan. As for all scans without a given alignment, the pose of the scan will again be calculated automatically.

ServerPostAlignScan

- “id”: NonnegativeInteger64
- “pose”: Pose2D
- “points”: ServerPostAlignScanPoints
- “alignType”: ServerPostAlignType

Describes a scan with an “id”, a “pose”, and an “alignType”. The scan consists of a number of laser scan “points”. These points are given in meters relative to the “pose”. The “alignType” specifies how the scan was aligned.

ServerPostAlignScanPoints

- “x”: Array of IEEE754Double (in meters)
- “y”: Array of IEEE754Double (in meters)

Contains the positions of the laser scan points that make up a laser scan. The arrays “x” and “y” contain the Cartesian x and y coordinates of the scan points. Coordinates are given relative to the scan pose in meters. The two arrays have the same length.

Two elements from “x” and “y” with the same index specify the coordinate of a single scan point. For example, the first value from “x” and the first value from “y” together form the coordinate of the first point in the scan.

9.2.5 Types

ServerPostAlignType Describes how a given scan was aligned.

Label	Value	Description
UNALIGNED	0	The scan was not aligned. The scan’s pose was calculated automatically.
FIXED	1	The scan’s pose is fixed and cannot be aligned. Attempting to align this scan will result in an error.
POST_ALIGNED	2	The scan’s pose was aligned according to the values given in the ServerPostAlignMapMessage.
GLOBAL_ALIGNED	3	The scan’s pose was aligned when the map was first created using the ClientGlobalAlign module.
ABSOLUTE_POSE	4	The scan’s pose was affected by a measurement of the absolute pose sensor.

Media Type “application/AlignedScans” A Container of this media type holds multiple scans in a base64-encoded (see FAQ for an example) binary representation. The binary representations of the scans are densely packed within the container. Thus the representation of one scan is immediately followed by that of the next scan.

The binary representation of each scan follows the conventions for datagrams used by binary interfaces. After decoding the base64-encoded data, the binary representation can thus be interpreted in the same way as a series of datagrams sent by a binary interface:

Name	Size (B)	Type	Restriction
id	8	Integer64	-
pose	24	Pose2DDouble	-
alignType	8	Integer64	as ServerPostAlignType
points		Array	
–>length	8	UnsignedInteger64	-
–>elements	8	Position2DSingle	-

Each binary representation of this type describes a scan with an “id”, a “pose”, and an “alignType”. The scan consists of a number of laser scan “points”. These points are given in meters relative to the “pose”. The “alignType” specifies how the scan was aligned.

9.3 SERVER-SIDE INTERNAL SETTINGS (MODULE: *SERVERINTERNAL*)

The ServerInternal module has been **deprecated** and will be removed in the next minor release version. Use the equivalent methods, messages, objects, and types from the Internal module instead. In the meantime, the existing methods, messages, objects, and types have been preserved for compatibility. However, these are only aliases for their counterparts from the Internal module.

9.3.1 serverInternalFleetConfigSet

Deprecated, alias for internalUserDataSet.

- QueryType: ServerInternalFleetConfigSettingsMessage
- ResponseType: ServerInternalResponseMessage

Query Value

- sessionId: the session setting the configuration
- the new ROKIT Locator Fleet management settings

Response Value

- response code as defined in the relevant section

9.3.2 serverInternalFleetConfigGet

Deprecated, alias for internalUserDataGet.

- QueryType: ServerInternalQueryMessage
- ResponseType: ServerInternalFleetConfigSettingsResponseMessage

Query Value

- sessionId: the session retrieving the configuration

Response Value

- response code as defined in the relevant section
- the stored ROKIT Locator Fleet management settings, or an empty string if an error occurred

9.3.3 Messages

ServerInternalQueryMessage **Deprecated**, alias for InternalQueryMessage.

ServerInternalResponseMessage **Deprecated**, alias for InternalReponseMessage.

ServerInternalFleetConfigSettingsMessage **Deprecated**, alias for InternalUserDataSettingsMessage.

ServerInternalFleetConfigSettingsResponseMessage **Deprecated**, alias for InternalUserDataSettingsResponseMessage.

9.3.4 Objects

ServerInternalFleetConfigSettings **Deprecated**, alias for InternalUserDataSettings.

9.4 SERVER-SIDE USER ACCOUNT MANAGEMENT (MODULE: *SERVERUSER*)

The ServerUser module has been **deprecated** and will be removed in the next minor release version. Use the equivalent methods, messages, objects, and types from the User module instead. In the meantime, the existing methods, messages, objects, and types have been preserved for compatibility. However, these are only aliases for their counterparts from the User module.

9.4.1 serverUserAdd

Deprecated, alias for userAdd.

- QueryType: ServerUserAddMessage
- ResponseType: ServerUserResponseMessage

9.4.2 serverUserDelete

Deprecated, alias for userDelete.

- QueryType: ServerUserDeleteMessage
- ResponseType: ServerUserResponseMessage

9.4.3 serverUserChangePassword

Deprecated, alias for userChangePassword.

- QueryType: ServerUserPasswordChangeMessage
- ResponseType: ServerUserResponseMessage

9.4.4 serverUserChangeOwnPassword

Deprecated, alias for userChangeOwnPassword.

- QueryType: ServerUserOwnPasswordChangeMessage
- ResponseType: ServerUserResponseMessage

9.4.5 serverUserList

Deprecated, alias for userList.

- QueryType: ServerUserQueryMessage
- ResponseType: ServerUserListResponseMessage

9.4.6 serverUserListGroup

Deprecated, alias for userListGroup.

- QueryType: ServerUserQueryMessage
- ResponseType: ServerUserGroupListResponseMessage

9.4.7 Messages

ServerUserAddMessage

Deprecated, alias for UserAddMessage.

ServerUserDeleteMessage

Deprecated, alias for UserDeleteMessage.

ServerUserPasswordChangeMessage

Deprecated, alias for UserPasswordChangeMessage.

ServerUserOwnPasswordChangeMessage

Deprecated, alias for UserOwnPasswordChangeMessage.

ServerUserResponseMessage

Deprecated, alias for UserResponseMessage.

ServerUserQueryMessage

Deprecated, alias for UserQueryMessage.

ServerUserListResponseMessage

Deprecated, alias for UserListResponseMessage.

ServerUserGroupListResponseMessage

Deprecated, alias for UserGroupListResponseMessage.

9.4.8 Objects

ServerUserInfo

Deprecated, alias for UserInfo.

9.4.9 Types

ServerUserGroup

Deprecated, alias for UserGroup.

10 Common Support – RPC Methods

10.1 SUPPORT REPORTS (MODULE: *SUPPORTREPORT*)

10.1.1 supportReportCreate

Creates a new ROKIT Locator Support Report that documents the current ROKIT Locator state.

The report is added to a report list, from which it can be retrieved later. This call blocks until the creation of the ROKIT Locator Support Report has finished.

Note that calling supportReportList during the write operation involved in the creation of the ROKIT Locator Support Report (i.e. while supportReportCreate is still working) displays a path to the ROKIT Locator Support Report file but the file might still be written to.

An optional start and end time can be specified to limit the amount of data that is included in the report. The start and end time must lie within the timespan covered by the driveDataDuration. This timespan extends from the current time up to `driveDataDurationHours` hours into the past. If the chosen start or end time lie outside of this maximum timespan, or if the start time lies at or after the end time, a `SUPPORTREPORT_TIMESPAN_INVALID` error may result. In this case, no report will be created, and the maximum possible start and end time will be given in a diagnostic message.

Note: The start and end time are given relative to the clock of the system on which the ROKIT Locator is running. When in doubt, the start and end time should be omitted to ensure that all available data is included in the report.

- QueryType: SupportReportTimespanMessage
- ResponseType: SupportReportNameResponseMessage

Query Value

- sessionId: the session requesting the report
- timeStart (optional, default value: invalid): the time after which support data should be included. If this timestamp is omitted or set to invalid, the support data included in the report will begin at the earliest possible time.
- timeEnd (optional, default value: invalid): the time until which support data should be included. If this timestamp is omitted or set to invalid, the support data included in the report will end at the latest possible time.
-

Response Value

- response code as defined in the relevant section
- name of the newly created ROKIT Locator Support Report (empty string on error)

10.1.2 supportReportCreateMinimal

Creates a new minimal ROKIT Locator 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. This call blocks until the creation of the ROKIT Locator Support Report has finished.

Note that calling supportReportList during the write operation involved in the creation of the ROKIT Locator Support Report (i.e. :while supportReportCreateMinimal is still working) would display a path to the ROKIT Locator Support Report file but it might still be written to.

- 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 ROKIT Locator Support Report (empty string on error)

10.1.3 supportReportSetDescription

Stores a ROKIT Locator 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 ROKIT Locator 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 ROKIT Locator Support Report

Response Value

- response code as defined in the relevant section

10.1.4 supportReportList

Retrieves a list of all known ROKIT Locator 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 ROKIT Locator Support Reports

10.1.5 supportReportGetPath

Retrieves the specified ROKIT Locator 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 resource from which the ROKIT Locator 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`

10.1.6 supportReportDelete

Deletes the specified ROKIT Locator 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

10.1.7 Messages

SupportReportQueryMessage

- “sessionId”: SessionId

SupportReportNameResponseMessage

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

SupportReportTimespanMessage

- “sessionId”: SessionId
- “timeStart”: Timestamp (optional)
- “timeEnd”: Timestamp (optional)

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

10.1.8 Objects

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

10.2 SYSTEM BACKUP, UPDATE, RECOVERY (MODULE: *SUPPORTRECOVERY*)

10.2.1 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

10.2.2 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)

10.2.3 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

10.2.4 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

10.2.5 supportRecoveryFrom

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

It is required to restart the docker container after a successful call to this method because some changes made by a recovery can only be applied during container startup.

- 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

10.2.6 supportRecoveryExport

Exports an existing recovery point into a Data Exchange Archive. This archive can be used together with the Data Exchange module to transfer the recovery point to another compatible ROKIT Locator. It can also be used to restore the recovery point at a later point. This method does not modify the target recovery point in any way.

- QueryType: SupportRecoveryNameMessage
- ResponseType: SupportRecoveryArchiveNameResponseMessage

Query Value

- sessionId: the session exporting the recovery point
- recoveryName: the name of the recovery point that should be exported

Response Value

- responseCode: response code as defined in the relevant section
- archiveName: the name of the archive in which the exported recovery point was stored. May be empty in case of an error

10.2.7 supportRecoveryImport

Imports a recovery point from a Data Exchange Archive. Suitable archives can be created by calling supportRecoveryExport. They can also be transferred from a compatible ROKIT Locator by using the Data Exchange module. This method does not modify the target archive in any way.

If successful, the recovery point stored within the archive is loaded into the ROKIT Locator and can immediately be used just like any other recovery point. The name of the recovery point imported in this manner is returned as part of the response.

If a recovery point of the same name already exists, an `ENTITY_ALREADY_EXISTS` error will result. In this case, the response contains the name of the existing recovery point that is preventing the archive from being imported. To import the archive, delete the existing recovery point.

Note that recovery points may only be imported from archives that were created by compatible versions of the ROKIT Locator. Attempting to import a recovery point from an archive that was created by an incompatible version of the ROKIT Locator will result in a `SUPPORTRECOVERY_ARCHIVE_INCOMPATIBLE` error. In general, archives are considered incompatible if they were created by a ROKIT Locator with a newer (higher) version than the version of the ROKIT Locator currently being used. In this case, the ROKIT Locator currently being used will have to be upgraded to a compatible version before the archive can be imported.

Attempting to import a ROKIT Locator Client recovery point from an archive that contains another kind of entity (such as a ROKIT Locator Server recovery point) will result in a `SUPPORTRECOVERY_ARCHIVE_WRONG_TYPE` error.

Archive files are designed to detect inadvertent changes or data corruption. If the archive has been changed since it was exported, a `SUPPORTRECOVERY_ARCHIVE_INVALID` may result. This ensures that a recovery point that is imported from an archive has not been damaged or modified after it was exported. If the archive file has been severely damaged and cannot be read at all, a `FILE_ACCESS_FAILED` error may be returned instead.

When a new autosave is imported and the configured maximum number of autosaves is exceeded, then the oldest autosave will be deleted on the next auto removal run. The age of an autosave is determined by its write time to the disk.

Note that the ROKIT Locator itself never modifies the contents of an archive after it has been written. Therefore this kind of error is likely the result of problems in data storage or transmission that lie outside of the ROKIT Locator.

- QueryType: SupportRecoveryArchiveNameMessage
- ResponseType: SupportRecoveryNameResponseMessage

Query Value

- sessionId: the session importing the recovery point
- archiveName: the name of the archive from which the recovery point should be imported

Response Value

- responseCode: response code as defined in the relevant section
- recoveryName: the name of the recovery point that was imported from the archive. May be empty in case of an error

10.2.8 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

SupportRecoveryArchiveNameMessage

- “sessionId”: SessionId
- “archiveName”: DataExchangeArchiveName

SupportRecoveryArchiveNameResponseMessage

- “responseCode”: ResponseCode
- “archiveName”: DataExchangeArchiveName or EmptyString

10.2.9 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 (‘/’).

11 ROKIT Locator Server Support – RPC Methods

11.1 ROKIT LOCATOR SERVER SUPPORT FLEET INFO (MODULE: *SUPPORTFLEETINFO*)

11.1.1 supportFleetinfoCreate

Creates a new ROKIT Locator Server Support Fleet Info that documents the current ROKIT Locator fleet state. The fleet info is added to a fleet info list, from which it can be retrieved later. This call blocks until the creation of the ROKIT Locator Server Support Fleet Info has finished.

Note that calling supportFleetinfoList during the write operation involved in the creation of the ROKIT Locator Server Support Fleet Info (i.e. while supportFleetinfoCreate is still working) displays a path to the ROKIT Locator Server Support Fleet Info file but the file might still be written to.

- QueryType: SupportFleetinfoQueryMessage
- ResponseType: SupportFleetinfoNameResponseMessage

Query Value

- sessionId: the session requesting the fleet info

Response Value

- response code as defined in the relevant section
- name of the newly created ROKIT Locator Server Support Fleet Info (empty string on error)

11.1.2 supportFleetinfoSetDescription

Stores a ROKIT Locator Server Support Fleet Info description, which can include additional support information supplied by a user. This description will be stored in a buffer and included in the next ROKIT Locator Server Support Fleet Infos created under this session. Calling this method repeatedly without creating a fleet info will overwrite the previous description buffer.

- QueryType: SupportFleetinfoDescriptionMessage
- ResponseType: SupportFleetinfoResponseMessage

Query Value

- sessionId: the session for which the description should be set
- fleetinfoDescription: the text which should be added to the next ROKIT Locator Server Support Fleet Info

Response Value

- response code as defined in the relevant section

11.1.3 supportFleetinfoList

Retrieves a list of all known ROKIT Locator Server Support Fleet Infos.

- QueryType: SupportFleetinfoQueryMessage
- ResponseType: SupportFleetinfoListResponseMessage

Query Value

- sessionId: the session requesting the list

Response Value

- response code as defined in the relevant section
- the names of all known ROKIT Locator Server Support Fleet Infos

11.1.4 supportFleetinfoGetPath

Retrieves the path to the ROKIT Locator Server Support Fleet Info.

- QueryType: SupportFleetinfoNameMessage
- ResponseType: SupportFleetinfoNameUrlResponseMessage

Query Value

- sessionId: the session requesting the fleet info
- the name of the fleet info 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 resource from which the ROKIT Locator Server Support Fleet Info 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/fleetinfo/20230510T143600Z-fleetinfo.tar.bz2?token=99a27474-1eeb-42bd-b7eb-6a2d92661c98`

11.1.5 supportFleetinfoDelete

Deletes the specified ROKIT Locator Server Support Fleet Info.

- QueryType: SupportFleetinfoNameMessage
- ResponseType: SupportFleetinfoListResponseMessage

Query Value

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

Response Value

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

11.1.6 Messages

SupportFleetinfoQueryMessage

- “sessionId”: SessionId

SupportFleetinfoNameResponseMessage

- “responseCode”: ResponseCode
- “fleetinfoName”: SupportFleetinfoName or EmptyString

SupportFleetinfoDescriptionMessage

- “sessionId”: SessionId
- “fleetinfoDescription”: string

SupportFleetinfoListResponseMessage

- “responseCode”: ResponseCode
- “fleetinfoNames”: array of SupportFleetinfoName

SupportFleetinfoNameMessage

- “sessionId”: SessionId
- “fleetinfoName”: SupportFleetinfoName

SupportFleetinfoResponseMessage

- “responseCode”: ResponseCode

SupportFleetinfoNameUrlResponseMessage

- “responseCode”: ResponseCode
- “fleetinfoNameUrl”: SupportFleetinfoNameUrl or EmptyString

11.1.7 Objects

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

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

12 Configuration Entries

The following section lists all configuration entries which can be used to configure the ROKIT Locator.

Some configuration entries are grouped using the notation `{...}`, e. g. `example.{a,b,c}`. This means that there are the configuration parameters `example.a`, `example.b`, `example.c`.
Are there multiple such `{...}` groups in one configuration entry description, the cross product of all combination exists, e. g. for `example2.{a,b}.dummy.{c,d}` all four parameters `example2.a.dummy.c`, `example2.a.dummy.d`, `example2.b.dummy.c`, `example2.a.dummy.d`.

12.1 COMMON CONFIGURATION ENTRIES

The configuration entries listed in the following table are common to various Rexroth ROKIT products and components (if product consists of multiple parts). Please be aware that different products may have different versions of the common API modules implemented and therefore the configuration entries may be different.

12.1.1 Certificate Management Entries (Module: *Certificate*)

Configuration Parameter	Type	Permission
	Description	
Certificate.enableRevocationList	boolean	admin
	Determines whether certificates used in secure communication are checked against the provided certificate revocation list (CRL). Only takes effect after restarting the ROKIT Locator.	

12.1.2 Software Licensing Entries (Module: *Licensing*)

Configuration Parameter	Type	Permission
	Description	
Licensing.getServedLicense Automatically	boolean	user, admin
	Automatically request a new license from the Revenera License Server at startup or on config value change, if not already present. It is recommended to set this parameter to false, in order to avoid blocking multiple licenses when the Host ID changes.	
Licensing.licensingMethod	AsciiCharacterString	user, admin
	Method used to determine the systems Host ID (possible values see chapter 15.1).	

(table continued)

Configuration Parameter	Type	Permission
	Description	
Licensing.localServerUrl	HttpsUrl or EmptyString	user, admin
	Request a license from the Revenera License Server at this url. The url should be similar to `https://some-host:11443/request`, when using a standard installation of the Revenera License Server.	
Licensing.preferredDongleSerial	AsciiCharacterString	user, admin
	Use the given dongle serial number as the Host ID, especially if multiple dongles are available.	

12.1.3 Backup, Recovery Entries (Module: *SupportRecovery*)

Configuration Parameter	Type	Permission
	Description	
SupportRecovery.autosaveIntervalDays	NonnegativeInteger64	admin
	Time interval between two autosaves (in days); must be larger than 0	
SupportRecovery.enableAutosave	boolean	admin
	Enable automatic creation of recovery points (autosave).	
SupportRecovery.numberOfAutosaves	NonnegativeInteger64	admin
	Number of autosave points to keep; if zero all autosaves are removed.	

12.1.4 Support Reports Entries (Module: *SupportReport*)

Configuration Parameter	Type	Permission
	Description	
SupportReport.driveDataDurationHours	NonnegativeInteger64	admin
	Duration of recording data for ROKIT Locator Support Reports (in hours).	
SupportReport.recordDriveData	boolean	admin
	Enable data recording for ROKIT Locator Support Reports.	

12.2 ROKIT LOCATOR CLIENT CONFIGURATION ENTRIES

The following table lists all configuration entries of the ROKIT Locator Client.

12.2.1 Application Management Entries (Module: *ClientApplication*)

Configuration Parameter	Type	Permission
	Description	
ClientApplication.mapServerAddress	NetworkAddress	user, admin
	Address (ROKIT Locator Client resolvable netname of IP address) of the ROKIT Locator Server, without a port.	
ClientApplication.mapServerHttpProxyUrl	EmptyString or NetworkAddress	user, admin
	URL of the HTTP Proxy, which the ROKIT Locator Client uses to create an HTTP tunnel to the ROKIT Locator Server	
ClientApplication.mapServerHttpProxyToken	BasicAuth, BearerToken or EmptyString	user, admin
	Access token for the HTTP Proxy	
ClientApplication.mapServerPortsStart	Port	user, admin
	The port under which the ROKIT Locator Server can be reached. 21638 by default	
ClientApplication.vehicleName	AsciiCharacterString	user, admin
	Vehicle name that is used to recognize specific vehicles in the ROKIT Locator Server logs. Changing this configuration parameter will invalidate a served license without returning it.	

12.2.2 Laser Masking Entries (Module: *ClientLaserMask*)

Configuration Parameter	Type	Permission
	Description	
ClientLaserMask.{laser,laser2,laser3,laser4,laser5,laser6}		
ClientLaserMask.{laser,laser2,laser3,laser4,laser5,laser6}.maxRange	IEEE754Double	user, admin
	Laser ranges above this value (in meters) will be ignored; must be greater than 0.	
ClientLaserMask.{laser,laser2,laser3,laser4,laser5,laser6}.maxRangeLines	array of IEEE754Double	user, admin
	Each group of four consecutive values (x1, y1, x2, y2, ...) specified the start and end points of a line relative to the first laser scanner (in meters); laser beams intersecting these lines will be ignored if the measured range is *above* the distance to the intersection point.	
ClientLaserMask.{laser,laser2,laser3,laser4,laser5,laser6}.minRange	IEEE754Double	user, admin
	Laser ranges below this value (in meters) will be ignored; must be greater than or equal to 0.	

(table continued)

Configuration Parameter	Type	Permission
	Description	
ClientLaserMask. {laser,laser2,laser3,laser4,laser5,laser6}. minRangeLines	array of IEEE754Double	user, admin
	Each group of four consecutive values (x1, y1, x2, y2, ...) specified the start and end points of a line relative to the first laser scanner (in meters); laser beams intersecting these lines will be ignored if the measured range is *below* the distance to the intersection point.	

12.2.3 Localization Entries (Module: *ClientLocalization*)

Configuration Parameter	Type	Permission
	Description	
ClientLocalization.activeMapName	ServerMapName	user, admin
	Name of the map to use on this ROKIT Locator Client; must match the name of a map on the ROKIT Locator Server.	
ClientLocalization.allowRelocalization	boolean	user, admin
	If true, the ROKIT Locator Client can relocalize after the localization is considered to be lost, even if this causes a jump in the vehicle pose. If false, the estimated pose will not jump to relocalize after the localization is considered to be lost.	
ClientLocalization.autostart	boolean	user, admin
	Determines whether localization begins immediately after starting the ROKIT Locator Client.	
ClientLocalization.mapHistoryDuration	NonnegativeInteger64	admin
	Timespan covered by the map history (in seconds).	
ClientLocalization.sendMapUpdates	boolean	admin
	Enable or disable sending map updates. Note that setting this to false does not disable receiving updated maps, e.g. when other clients are updating the same server map.	
ClientLocalization.storeMapHistory	boolean	admin
	Enable or disable storing a map history.	

12.2.4 Sensor Management Entries (Module: *ClientSensor*)

Configuration Parameter	Type	Permission
	Description	
ClientSensor.absolutePoseAddress	NetworkAddress	user, admin
	Address `<client resolvable netname of IP address>:<port>` of the external absolute pose source.	
ClientSensor.absolutePoseEncryption	boolean	user, admin
	Determines if TLS should be used when connecting to the external absolute pose source.	
ClientSensor.enableAbsolutePose	boolean	user, admin
	Determines if an external absolute pose sensor should be used.	
ClientSensor.enableImu	boolean	user, admin
	Determines if an external inertial measurement unit should be used.	
ClientSensor.enableLaser2	boolean	user, admin
	Determines if the dual laser feature should be used by enabling laser 2.	
ClientSensor.enableLaser{3,4,5,6}	boolean	user, admin
	Determines if the auxiliary laser feature should be used by enabling laser 3, 4, 5, or 6.	
ClientSensor.enableOdometry	boolean	user, admin
	Determines if an external odometry source should be used.	
ClientSensor.imuAddress	NetworkAddress	user, admin
	Address `<client resolvable netname of IP address>:<port>` of the external inertial measurement unit	
ClientSensor.imuEncryption	boolean	user, admin
	Determines if TLS should be used when connecting to the external IMU source.	
ClientSensor.odometryAddress	NetworkAddress	user, admin
	Address `<client resolvable netname of IP address>:<port>` of the external odometry source.	
ClientSensor.odometryEncryption	boolean	user, admin
	Determines if TLS should be used when connecting to the external odometry source.	
ClientSensor.vehicleErrorModel TransformImu.enabled	boolean	user, admin
	Enable the usage of these coordinates instead of those in ClientSensor.vehicleTransformImu if the center of rotation and the IMU reference frame do not match	

(table continued)

Configuration Parameter	Type	Permission
	Description	
ClientSensor.vehicleErrorModel TransformImu. {x,y,z,roll,pitch,yaw}	IEEE754Double, -100 ≤ x, y, z ≤ 100	user, admin
	Position and orientation (in meters and degrees) of the platform center of rotation with respect to the vehicle reference frame used in the IMU error model.	
ClientSensor.vehicleErrorModel TransformOdometry.enabled	boolean	user, admin
	Enable the usage of these coordinates instead of those in ClientSensor.vehicleTransformOdometry if the center of rotation and the vehicle reference frame do not match.	
ClientSensor.vehicleErrorModel TransformOdometry. {x,y,z,roll,pitch,yaw}	IEEE754Double, -100 ≤ x, y, z ≤ 100	user, admin
	Position and orientation (in meters and degrees) of the platform center of rotation with respect to the vehicle reference frame used in the odometry error model	
ClientSensor.vehicleTransform AbsolutePose.{x,y,z,roll,pitch,yaw}	IEEE754Double, -100 ≤ x, y, z ≤ 100	user, admin
	Position and orientation (in meters and degrees) of the origin of the absolute pose sensor's reference frame with respect to the vehicle reference frame.	
ClientSensor.vehicleTransformImu. {x,y,z,roll,pitch,yaw}	IEEE754Double, -100 ≤ x, y, z ≤ 100	user, admin
	Position and orientation (in meters and degrees) of the origin of the IMU's reference frame with respect to the vehicle reference frame.	
ClientSensor.vehicleTransform Odometry.{x,y,z,roll,pitch,yaw}	IEEE754Double, -100 ≤ x, y, z ≤ 100	user, admin
	Position and orientation (in meters and degrees) of the origin of the odometry unit's reference frame with respect to the vehicle reference frame.	
ClientSensor.enableReflectorMarkers	boolean	user, admin
	Determines whether reflector markers should be detected and used.	
ClientSensor.customReflectorParameters		
ClientSensor.customReflector Parameters.enabled	boolean	user, admin
	If true, customer-provided parameters are used for reflector marker detection. If false, pre-configured default parameters will be used and all customer-provided reflector parameters are ignored.	

(table continued)

Configuration Parameter	Type	Permission
	Description	
ClientSensor.customReflector Parameters.highIntensityThreshold	IEEE754Double -1 or ≥ 0	user, admin
	Intensity threshold for reflector marker detection. If set to -1, the laser scanner specific internal default value is used.	
ClientSensor.customReflector Parameters.minIntensityChangeFactor	IEEE754Double -1 or ≥ 0	user, admin
	Threshold for relative intensity between reflector marker and background. If set to -1, the laser scanner specific internal default value is used.	
ClientSensor.customReflector Parameters.minDistance	IEEE754Double -1 or ≥ 0	user, admin
	Minimum distance (in meters) below which reflector markers will not be detected. If set to -1, the internal default value is used.	
ClientSensor.customReflector Parameters.maxDistance	IEEE754Double -1 or > 0	user, admin
	Maximum distance (in meters) beyond which reflector markers will not be detected. If set to -1, the internal default value is used.	
ClientSensor.customReflector Parameters.windowSize	IEEE754Double -1 or > 0	user, admin
	Radius around potential reflective marker (in meters) which will be considered during detection. If set to -1, the internal default value is used.	
ClientSensor.customReflector Parameters.minWallLength	IEEE754Double -1 or > 0	user, admin
	Minimum length of a wall segment (in meters) carrying a potential reflector marker. If set to -1, the internal default value is used.	
ClientSensor.customReflector Parameters.minWallPoints	Integer64 -1 or > 0	user, admin
	Minimum number of laser scan points on a wall segment carrying a potential reflector marker. If set to -1, the internal default value is used.	
ClientSensor.customReflector Parameters.maxWallFitError	IEEE754Double -1 or > 0	user, admin
	Maximum error by which a wall segment carrying a potential reflector marker may deviate from a straight line. If set to -1, the internal default value is used.	

(table continued)

Configuration Parameter	Type	Permission
	Description	
ClientSensor.customReflector Parameters.minWallViewingAngle	IEEE754Double -1 or ≥ 0 and < 90	user, admin
	Minimum angle (in degrees) between the viewing direction and the wall surface at which a wall-mounted reflector marker may be detected. A value of 0 allows all viewing angles. If set to -1, the internal default value is used.	
ClientSensor.customReflector Parameters.maxWallViewingAngle	IEEE754Double -1 or > 0 and ≤ 90	user, admin
	Maximum angle (in degrees) between the viewing direction and the wall surface at which a wall-mounted reflector marker may be detected. A value of 90 allows all viewing angles. If set to -1, the internal default value is used.	
ClientSensor.customReflector Parameters.markerWidth	IEEE754Double -1 or > 0	user, admin
	Nominal width (in meters) of the reflector markers to be detected. If set to -1, the internal default value is used.	
ClientSensor.customReflector Parameters.markerWidthTolerance Factor	IEEE754Double -1 or > 0	user, admin
	Factor by which the measured marker width may exceed the specified marker width. A factor of 1.5 means that the measured marker width may be up to 50% higher than specified. If set to -1, the internal default value is used.	
ClientSensor.customReflector Parameters.expectedAdditionalMarker Points	Integer64 -1 or $>= 0$	user, admin
	Expected difference between the pre-computed and measured number of laser scan points that cover a detected marker. If set to -1, the internal default value is used.	
ClientSensor.customReflector Parameters.allowedMarkerPointsError	Integer64 -1 or $>= 0$	user, admin
	Allowed error between the expected and measured number of laser scan points that cover a detected marker. If set to -1, the internal default value is used.	
ClientSensor.{laser,laser2,laser3,laser4,laser5,laser6}		
ClientSensor. {laser,laser2,laser3,laser4,laser5,laser6}. address	NetworkAddress	user, admin
	Address of the laser in question. Refer to the ROKIT Locator User Manual for details on supported laser scanners.	

(table continued)

Configuration Parameter	Type / Description	Permission
ClientSensor.{laser,laser2,laser3,laser4,laser5,laser6}.clientAddress	IPv4Address Only when using sickcola2udp, sickpicoscanudp or keyenceszvudp as ClientSensor.laser.type: Address of the host network interface to which the laser should send data via UDP.	user, admin
ClientSensor.{laser,laser2,laser3,laser4,laser5,laser6}.mirrorLaserScans	boolean Flag that indicates if the laser is mounted upside down	user, admin
ClientSensor.{laser,laser2,laser3,laser4,laser5,laser6}.type	string Name of the laser driver to use: omron, sicklms, sickcola2, sickcola2udp, sickpicoscanudp, idec, pfr2000, keyenceszvudp, simple, simpletls. Refer to the ROKIT Locator User Manual for details on supported laser scanners.	user, admin
ClientSensor.{laser,laser2,laser3,laser4,laser5,laser6}.useIntensities	boolean Flag that indicates if the laser should use intensities.	user, admin
ClientSensor.{laser,laser2,laser3,laser4,laser5,laser6}.vehicleTransformLaser.{x,y,z,roll,pitch,yaw}	IEEE754Double, -100 $\leq$ x, y, z $\leq$ 100 Position and orientation (in meters and degrees) of the origin of the laser scanner's reference frame with respect to the vehicle reference frame.	user, admin
ClientSensor.sensorFusion		
ClientSensor.sensorFusion.absolutePoseOverride	boolean **Note:** This parameter currently has no effect and is always treated as false. If true, absolute pose data used for localization or mapping is trusted fully and will completely override the information from all other sensors during sensor fusion. If false, the absolute pose data is fused with the information from other sensors and may be overruled.	user, admin
ClientSensor.sensorFusion.useAbsolutePose	boolean Determines whether or not to use available absolute pose data during localization or mapping.	user, admin
ClientSensor.sensorFusion.useOdometry	boolean Determines whether or not to use available wheel odometry data during localization or mapping.	user, admin

The parameter `ClientSensor.laser.type` governs the primary laser driver to be used by the ROKIT Locator Client. Note that the format of the `ClientSensor.laser.address` 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 ROKIT Locator manual for details on supported laser scanners as well as their associated `ClientSensor.laser.type` and `ClientSensor.laser.address` settings. The parameters for the secondary laser are indicated with the prefix `ClientSensor.laser2` e.g. `ClientSensor.laser2.type`

The parameters with the prefix `ClientSensor.laser3`, `ClientSensor.laser4`, `ClientSensor.laser5`, or `ClientSensor.laser6` govern the auxiliary laser scanners 3-6. These auxiliary laser scanners are not used for most ROKIT Locator Client features. In particular, they have no effect on recording, mapping, map updates, or localization. However, the ROKIT Locator Client() will connect to these laser scanners and provide their scan data through the ClientSensorLaserOutput output binary interface and the clientSensorGetLaserScan API method.

12.3 ROKIT LOCATOR SERVER CONFIGURATION ENTRIES

The following table lists all configuration entries of the ROKIT Locator Server.

12.3.1 Server-side Map Management Entries (Module: *ServerMap*)

Configuration Parameter	Type	Permission
	Description	
ServerMap.parallelMapDownloads	NonnegativeInteger64 > 0, ≤ 16	admin
	Number of clients that can simultaneously download maps. Only takes effect after restarting the Server	
ServerMap.timeBetweenMapUpdates	NonnegativeInteger64 ≥ 60, ≤ 86400	user, admin
	Number of seconds that must pass before the ROKIT Locator Server publishes an updated map to connected clients. Note that newly-connected clients will always receive the most recent map.	

13 Binary Interfaces

Binary Interfaces handle low-latency, high-throughput transmission of homogeneous 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 appropriate section for a list of all Binary Interfaces and their default 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.

13.1 COMMON DEFINITIONS (SHARED ACROSS MODULES)

13.1.1 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 $[0, 1]$.

UnsignedInteger8 An unsigned integer of 8 bits with a possible value within the interval [0, 2^8) or [0, 255].

SignedInteger8 A two-complement signed integer of 8 bits with a possible value within the interval $[-2^7, 2^7)$ or [-128, 127].

UnsignedInteger16 A little-endian, unsigned integer of 16 bits with a possible value within the interval [0, 2^{16}) or [0, 65635].

SignedInteger16 A little-endian, two-complement signed integer of 16 bits with a possible value within the interval $[-2^{31}, 2^{31})$ or [-32768, 32767].

UnsignedInteger32 A 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].

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

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.

Position2DSingle A 2D position represented by Cartesian x/y coordinates (in meters).

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

Position2DDouble A 2D position represented by Cartesian x/y coordinates (in meters).

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

Position3DSingle A 3D position represented by Cartesian x/y/z coordinates (in meters).

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

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

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

Pose2DDouble A 2D pose represented by Cartesian x/y coordinates (in meters) and a yaw angle (in radians).

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

Polygon2DSingle A Polygon2DSingle is a closed line that must not self-intersect or contain any holes. It is defined as an array of points representing the vertices of the polygon, given in clockwise order. The polygon is closed, the first point must be numerically identical to the last point.

Name	Size (B)	Type	Restriction
elements		Array	-
–>length	4	UnsignedInteger32	-
–>elements	8 (each)	Position2DSingle	|x|, |y| <= 1.e+12

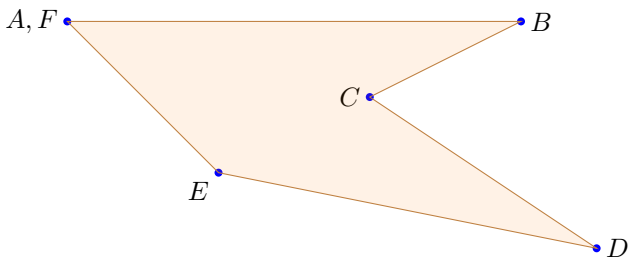

Figure 2: Example of a Polygon2DSingle object with 6 points. It is oriented in accordance with the clockwise directions, and its last point F is the same as the first point A.

PointCluster A PointCluster represents a cluster of points within a separately transmitted point cloud. A cluster of points is characterized by its position and orientation. This position and orientation is described as a prototype of type Pose2DDouble. For example, a cluster that contains points that belong to a physical object like a crate could be characterized by the position of the objects center and the orientation of one of its faces.

The cluster consists of all the points within the point cloud whose index lies within a given range [fromIndex, toIndex], as depicted in Figure 1. Therefore, only consecutive points can be part of a cluster and the point cloud may be reordered to achieve this.

Name	Size (B)	Type	Restriction
prototype	24	Pose2DDouble	yaw in $[-\pi_d, \pi_d]$
fromIndex	4	UnsignedInteger32	<= toIndex
toIndex	4	UnsignedInteger32	>= fromIndex

13.1.2 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 $|\pi - \pi_d|$.

13.1.3 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
length	4	UnsignedInteger32	<= 134217728, >= 0
elements	8 (each)	Position2DSingle	|x|,|y| <= 1.e+05, yaw in $[-\pi_f, \pi_f]$

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

13.1.4 Fixed-length Arrays

Definition An fixed-length array is a data structure that consists of a fixed number of objects of the same type. The fixed number of such elements within a given array is called the array's (fixed) length and will never change. Therefore the length is given in the API specification and – in contrast to normal arrays of section 13.1.3 – will *not* be transmitted with the actual object.

Datagram schema Note the following example:

Name	Size (B)	Type	Restriction
name of array	32	FixedArray[4]	
–>elements	8	Position2DSingle	|x|,|y| <= 1.e+05, yaw in $[-\pi_f, \pi_f]$

This is read as follows:

- the static length is given in the specification within the brackets [...] and the full size of the fixed-length array in the size column
- 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 4 elements have been transmitted

13.1.5 Extensions

Definition An extension is a datagram that is forward-compatible and back-expandable. It consists of a header region and an arbitrary number of ExtensionFields consisting of various datagrams.

The header region consists of an UnsignedInteger32 giving the size of the extension in bytes and an Array containing the offsets to all the ExtensionFields in the extension. This information allows the user to skip any part of the extension that is irrelevant for the task at hand.

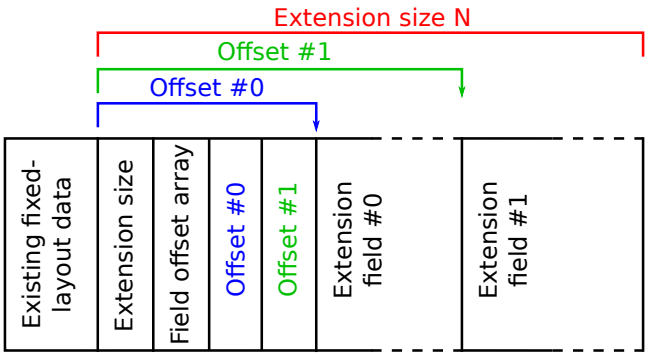

Figure 3: Schematic view on an extended datagram.

Figure 3 shows the structure of such an extension. It starts immediately after the original fixed-size part of the message with the size of the extension, followed by an array of the offsets of all individually addressable parts of the extension relative to the 0th byte of the extension in bytes.

Datagram schema Note the following example:

Name	Size (B)	Type	Restriction
extensionSize	4	UnsignedInteger32	No restrictions
fieldOffsetsArray		Array	
–>length	4	UnsignedInteger32	No restrictions
–>elements	4 (each)	UnsignedInteger32	No restrictions
Field 0: optionalPosition	8	Position2DSingle	|x|,|y| <= 1.e+05
Field 1: optionalArray	64	FixedLengthArray[4]	
–>elements	16 (each)	Position2DDouble	|x|,|y| <= 1.e+05

This is read as follows:

- The first item `extensionSize` is an UnsignedInteger32 giving the total size of the extension in bytes (including the 4 bytes of this field) – 88 bytes in this example.
- Next comes the `fieldOffsetsArray` which is an array of UnsignedInteger32. Its `length` contains the number of addressable ExtensionFields – two fields in this example. Its `elements` contain the offset of each ExtensionField from the 0th byte of the extension in bytes – 16 and 24 in this example.
- `ExtensionField 0` is the datagram at the position in the extension that is addressed by the 0th element of the fieldOffset array. In this example, it contains a single Position2DSingle of 8 bytes.
- `ExtensionField 1` is the datagram at the position in the extension that is addressed by the 1st element of the fieldOffset array. In this example it contains an fixed length array of Position2DDouble of 16 bytes each.

13.2 DIAGNOSTIC OUTPUT (MODULE: *DIAGNOSTIC*)

13.2.1 DiagnosticEvent Interface

This interface sends messages generated by the system. These messages correspond to the diagnostic entries returned by the diagnosticList API method.

Authentication is required to receive data from this interface. After connecting to the interface, users must send a DiagnosticAuthenticationDatagram. Users that do not send such a datagram will not receive any data from this interface. The DiagnosticAuthenticationDatagram must contain a valid session ID, as returned by a call to sessionLogin for the component in question. If a user sends more than one DiagnosticAuthenticationDatagram, the most recently sent datagram will be used for authentication. Sending any other data to this interface will result in the connection being closed.

All diagnostic messages are sent immediately to all connected users who have authenticated with a valid session ID, as described above. Users who have authenticated with an invalid session ID will not receive the contents of the diagnostic messages. Instead of the actual diagnostic message, such users will receive a DiagnosticEntryDatagram with a `responseCode` of `SESSION_INVALID`. Similarly, users that have sent a session ID that has expired will receive datagrams with a `responseCode` of `SESSION_EXPIRED`.

Diagnostic messages from the past are not stored, and cannot be retrieved using this interface. If the connection to the interface is interrupted, some diagnostic messages may thus not be received. To retrieve some or all previous diagnostic messages, call the diagnosticList API method.

DiagnosticAuthenticationDatagram

Name	Size (B)	Type	Restriction
sessionId		Array	
–>length	4	UnsignedInteger32	=36
–>elements	1	PrintableAsciiCharacter	No restrictions.

This datagram contains a session ID as a fixed-length array of PrintableAsciiCharacter. The content of this array are interpreted as a character string and must form a session ID.

DiagnosticEntryDatagram

Name	Size (B)	Type	Restriction
responseCode	8	SignedInteger64	No restrictions.
diagnosticEntry		Extension	
extSize	4	UnsignedInteger32	No restrictions.
fieldOffset		Array	
–>length	4	UnsignedInteger32	No restrictions.
–>elements	4	UnsignedInteger32	No restrictions.

Name	Size (B)	Type	Restriction
Field 0: id	8	SignedInteger64	No restrictions.
Field 1: componentName		Array	
–>length	4	UnsignedInteger32	<= 1024, >= 0
–>elements	1	PrintableAsciiCharacter	No restrictions.
Field 2: diagnosticCode	8	SignedInteger64	No restrictions.
Field 3: diagnosticGroup	8	SignedInteger64	one of: 0, 1, 2, 3, 4,
Field 4: timestamp	8	IEEE754Double	<= 1.e+12, >= 0.e+00
Field 5: diagnosticName		Array	
–>length	4	UnsignedInteger32	<= 1024, >= 0
–>elements	1 (each)	PrintableAsciiCharacter	No restrictions.
Field 6: diagnosticText		Array	
–>length	4	UnsignedInteger32	<= 65536, >= 0
–>elements	1 (each)	PrintableAsciiCharacter	No restrictions.
Field 7: additionalInfo		Array	
–>length	4	UnsignedInteger32	<= 1024, >= 0
–>elements		Array	
–> –>length	4	UnsignedInteger32	<= 1024, >= 0
–> –>elements	1 (each)	PrintableAsciiCharacter	No restrictions.

This datagram carries a single diagnostic message as well as a response code in binary form. The diagnostic message is encoded as an extension to allow for future modifications. This datagram is therefore equivalent to a DiagnosticArrayResponseMessage that contains a single DiagnosticEntry.

A `responseCode` with a value of `OK` indicates that the remaining content of the message is valid. All values within the datagram can then be interpreted as with the DiagnosticEntry. Other `responseCode` values indicate that an error has occured, and that the diagnostic entry carried by this datagram should be discarded.

Within this datagram, arrays of PrintableAsciiCharacter are used to represent the AsciiCharacterString from the DiagnosticEntry. Unlike an AsciiCharacterString, these arrays must be limited in their maximum length. These limits have been chosen generously and should be able to accomodate all diagnostic messages generated by the system. However, if a diagnostic entry would exceed the specified limits, it cannot be sent using this datagram. Instead, a datagram with a `responseCode` of `INVALID_MESSAGE_CONTENT` will be sent. The offending DiagnosticEntry can then be retrieved by calling the diagnosticList API method.

13.2.2 DiagnosticStatusMonitoring Interface

This interface periodically sends messages providing status information defined as StatusMonitoringState and further textual information about a number of system functionalities listed in the DiagnosticStatusMonitoringDatagram.

StatusMonitoringState

Label	Value (SignedInteger32)	Description
OK	1	Component function is ok.
NOT_APPLICABLE	2	Component does not support this function.
WARNING	4	Component may have restricted function.
ERROR	8	Component has an error.
INFO	16	Component provides the message as information.

DiagnosticStatusMonitoringDatagram

Name	Size (B)	Type	Restriction
–>length	4	SignedInteger32	No restrictions
–>elements	1 (each)	PrintableAsciiCharacter	JSON formatted string

13.3 SYSTEM (MODULE: *SYSTEM*)

The API Module System contains the following binary interfaces:

SystemStateHeartbeat enables an heartbeat to check the current status of ROKIT Locator overall health.

SystemJsonRequestIdList gives access to the currently running JSON-RPC requests of the ROKIT Locator.

In the following sections these interfaces are detailed.

13.3.1 SystemStateHeartbeat Interface

The SystemStateHeartbeat Binary Interface sends the current system state of the corresponding component. The heartbeat is send every second and additionally if the state changes. The interface uses the SystemStateHeartbeatDatagram. System is a common module, as a result the SystemStateHeartbeat Binary Interface is available on different ports for the ROKIT Locator and the support component. The assigned ports are listed in the FAQ.

SystemStateHeartbeatDatagram

Name	Size (B)	Type	Restriction
state	4	SignedInteger32	one of: 0, 1, 2, 3, 4,

- **state**: The system state.

SystemState

Label	Value (Signed-Integer32)	Description
STARTUP	0	Component is starting.
RUNNING	1	Component is running.
SIGNAL_SHUTDOWN	2	Component received signal to shutdown and is shutting down
SYSTEM_SHUTDOWN	3	Component received request to shutdown and is shutting down.
OUTOFORDER	4	Component is out of order.

13.3.2 SystemJsonRequestIdList Interface

The SystemJsonRequestIdList Binary Interface sends a list of the IDs of the currently ongoing JSON-RPC requests. The current SystemJsonRequestIdList is sent when connecting to the interface and whenever a JSON-RPC request is called or finished. If multiple JSON-RPC requests have the same ID, then the same ID will appear multiple times, once for each request with that ID. The interface uses the SystemJsonRequestIdListDatagram. The SystemJsonRequestIdList Binary Interface is available on different ports for the ROKIT Locator. The assigned ports are listed in the FAQ.

SystemJsonRequestIdListDatagram

Name	Size (B)	Type	Restriction
JsonRequestIdListExtension		Extension	
extensionSize	4	UnsignedInteger32	No restrictions
fieldOffset		Array	
–>length	4	UnsignedInteger32	No restrictions
–>elements	4	UnsignedInteger32	No restrictions
Field 0: idListEntries		Array	
–>num_entries	4	UnsignedInteger32	<= 8192, >= 0
–>entry_length	4	UnsignedInteger32	<= 8192, >= 0
–>entry_elements	1	PrintableAsciiCharacter	No restrictions.

13.4 CONTROL OUTPUT (MODULE: *CLIENTCONTROL*)

13.4.1 ClientControlMode Interface

The ClientControlMode Binary Interface provides information about the current operating mode of the ROKIT Locator Client. Specifically, it regularly transmits the mode as defined in Client Control Mode, using the ClientControlModeDatagram.

ClientControlModeDatagram

Name	Size (B)	Type	Restriction
controlMode	4	UnsignedInteger32	

The ROKIT Locator 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	Mode
2-0	LASEROUTPUT
5-3	ALIGN
8-6	REC
11-9	LOC
14-12	MAP
17-15	VISUALRECORDING
20-18	EXPANDMAP
31-21	Unused

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 ROKIT Locator 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

13.5 LOCALIZATION OUTPUT (MODULE: *CLIENTLOCALIZATION*)

The API Module ClientLocalization contains the following binary interfaces:

ClientLocalizationPose contains the main output of the ROKIT Locator, the localization result and status.

ClientLocalizationVisualization contains additional localization information for visualization.

ClientLocalizationMap contains the map data currently used for localization.

In the following sections these interfaces are detailed.

13.5.1 ClientLocalizationPose Interface

The ClientLocalizationPose interface provides continuous absolute and relative 2D planar poses. It operates while the ROKIT Locator 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.

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,
errorFlags	8	UnsignedInteger64	No restrictions.
infoFlags	8	UnsignedInteger64	No restrictions.
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.
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

Name	Size (B)	Type	Restriction
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 ROKIT Locator 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.
- **errorFlags**: A 64-bit bit field which indicates errors that may affect the localization.
- **infoFlags**: A 64-bit bit field which provides detailed information about specific aspects of the localization.
- **poseX**, **poseY**, **poseYaw**: The estimated Cartesian (x,y)-coordinates (in meters) and yaw angle (in radians) of the vehicle, 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 ROKIT Locator. If the ROKIT Locator 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 orientation given as quaternions. Note that the ROKIT Locator 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 (in meters) and yaw angle (in radians) of the sensor, given in an arbitrary, relative reference frame.

13.5.2 ClientLocalizationVisualization Interface

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

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
sensorOffsets		Array	
–>length	4	UnsignedInteger32	<= 2, >= 0
–>elements	8	UnsignedInteger64	<= 20000000
hasIntensities	1	Boolean	No restrictions
minIntensity	4	IEEE754Single	No restrictions
maxIntensity	4	IEEE754Single	No restrictions
intensities		Array	
–>length	4	UnsignedInteger32	<= 20000000, >= 0
–>elements	4	IEEE754Single	No restrictions
optionalData		Extension	
extensionSize	4	UnsignedInteger32	No restrictions
fieldOffset		Array	
–>length	4	UnsignedInteger32	No restrictions
–>elements	4	UnsignedInteger32	No restrictions
Field 0: reflectors		Array	
–>length	4	UnsignedInteger32	<= 20000000 >= 0
–>elements	32	PointCluster	0 <= from <= to <= 19999999; |x|,|y| <= 1.e+12

- **timestamp**: The time at which the ROKIT Locator 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 (in meters) and yaw angle (in radians) of the vehicle, 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. The (x,y) coordinates of each point are given in meters.
- **sensorOffsets**: Array of indices of the point cloud field **scan**. The i-th index represents the first scan point contributed by the i-th laser sensor. In interval notation the points that belong to the i-th sensor have indices within the interval [sensorOffsets[i], sensorOffsets[i+1]). If i is the index of the last sensor, it contributes the points with the indices [sensorOffsets[i], **scan**–>length).
- **hasIntensities**: Boolean indicating whether intensities are available. When set to false, the minIntensity and maxIntensity field are undefined and the intensities array has a length of zero.
- **minIntensity**: IEEE754Single indicating the minimum value for the entries in the intensities array.
- **maxIntensity**: IEEE754Single indicating the maximum value for the entries in the intensities array.
- **intensities**: Array of IEEE754Single containing the intensity values for the laser scan. Contains either the same number of elements as the scan, or zero elements.
- **optionalData**: Start of the Extension of this datagram. Provides its own size to enable the user to skip it easily.
- **reflectors**: Array of PointCluster. Each cluster defines points within the **scan** point cloud that were classified as a single reflector marker. Its prototype describes the estimated position (in meters) and normal angle (in radians) of the reflector.

13.5.3 ClientLocalizationMap Interface

The ClientLocalizationMap provides the map that is currently used for self-localization. It operates while the ROKIT Locator Client is in localization mode and uses the ClientLocalizationMapDatagram.

ClientLocalizationMapDatagram

Name	Size (B)	Type	Restriction
map		Array	
–>length	4	UnsignedInteger32	<= 134217728, >= 0
–>elements	8	Position2DSingle	|x|,|y| <= 1.e+05
optionalData		Extension	
extensionSize	4	UnsignedInteger32	No restrictions
fieldOffset		Array	
–>length	4	UnsignedInteger32	No restrictions
–>elements	4	UnsignedInteger32	No restrictions

Name	Size (B)	Type	Restriction
Field 0: reflectors		Array	
–>length	4	UnsignedInteger32	<= 134217728, >= 0
–>elements	4	PointCluster	0 <= from <= to <= 134217727; |x|,|y| <= 1.e+12

- **map**: A 2D point cloud representing the currently used laser map of the localization. The (x,y) coordinates of each point are given in meters.
- **optionalData**: Start of the Extension of this datagram. Provides its own size to enable the user to skip it easily.
- **reflectors**: Array of PointCluster. Each cluster defines points within the **map** point cloud that were classified as a single reflector marker. Its prototype describes the estimated position (in meters) and normal angle (in radians) of the reflector.

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 ROKIT Locator Client has not yet received a map from the ROKIT Locator Server
NOT_LOCALIZED	-1	Not localized, ROKIT Locator 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 successfully matched to the map
LOCALIZED_INTERNALUSE1	3	Localized, laser scans are successfully matched to the map. For internal use only, should be considered equivalent to LOCALIZED
LOCALIZED_INTERNALUSE2	4	Localized, laser scans are successfully matched to the map. For internal use only, should be considered equivalent to LOCALIZED

Localization Error Flags

Each defined bit within this bit field indicates an error that may negatively affect the localization. **Note:** Bits not listed in the following table are reserved for internal use and should be ignored.

If a given error has affected the current localization result, the corresponding bit is set to 1. Otherwise, the bit is 0. For the bit indices given in the table, an index of 0 describes the least significant bit within the 64-bit bit field.

Note that the localization result may still be usable even in case of an error. For example, the failure of a single sensor may have been compensated for by other sensors that function normally.

Bit	Label	Bitmask	Description
0	CLIENTSENSOR_ LASER_MISSING	0x0000 0000 0000 0001	No sensor data was available for the first laser
1	CLIENTSENSOR_ LASER2_MISSING	0x0000 0000 0000 0002	Dual-laser mode is enabled but no sensor data was available for the second laser
2	CLIENTSENSOR_ WHEEL_ODOMETRY_ MISSING	0x0000 0000 0000 0004	Wheel odometry is enabled and should be used, but no wheel odometry sensor data was available
3	CLIENTSENSOR_ ABSOLUTE_POSE_ MISSING	0x0000 0000 0000 0008	Absolute pose data is enabled and should be used, but no absolute pose sensor data was available
8	CLIENTLOCALIZATION_ NO_VALID_MAP	0x0000 0000 0000 0100	No valid map is available: The map is either missing or empty. To be expected while the localization state is `NOT_LOCALIZED_STARTUP` during initial start-up.

Localization Info Flags

Each defined bit within this bit field provides detailed information about the state of the localization. **Note:** Bits not listed in the following table are reserved for internal use and should be ignored.

If the state described by a given bit has occurred for the current localization result, the corresponding bit is set to 1. Otherwise, the bit is 0. For the bit indices given in the table, an index of 0 describes the least significant bit within the 64-bit bit field.

Bit	Label	Bitmask	Description
0	CLIENTSENSOR_ LASER_AVAILABLE	0x0000 0000 0000 0001	Sensor data from the first laser was available for localization

Bit	Label	Bitmask	Description
1	CLIENTSENSOR_ LASER2_AVAILABLE	0x0000 0000 0000 0002	Sensor data from the second laser was available for localization
2	CLIENTSENSOR_ WHEEL_ODOMETRY_ AVAILABLE	0x0000 0000 0000 0004	Sensor data from the wheel odometry was available for localization
3	CLIENTSENSOR_ ABSOLUTE_POSE_ AVAILABLE	0x0000 0000 0000 0008	Sensor data from the absolute pose sensor was available for localization
8	CLIENTLOCALIZATION_ MAP_UPDATE_ POSSIBLE	0x0000 0000 0000 0100	The localization was reliable enough that map updates are possible, if needed
9	CLIENTLOCALIZATION_ MAP_UPDATE	0x0000 0000 0000 0200	The ROKIT Locator Client recently generated a map update
13	CLIENTLOCALIZATION_ DOCKING_MODE	0x0000 0000 0000 2000	The ROKIT Locator Client is operating in docking mode

13.6 VISUAL RECORDING OUTPUT (MODULE: *CLIENTRECORDING*)

The API Module ClientRecording contains the following binary interfaces:

ClientRecordingVisualization contains information to visualize the current recording status.

ClientRecordingMap provides the current map created during the recording.

In the following sections these interfaces are detailed.

13.6.1 ClientRecordingVisualization Interface

The ClientRecordingVisualization interface provides information needed to visualize and evaluate the current recording process. It operates while the ROKIT Locator 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.

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

Name	Size (B)	Type	Restriction
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, yaw in $[-\pi_f, \pi_f]$
pathTypes		Array	
–>length	4	UnsignedInteger32	<= 20000000, >= 0
–>elements	4	SignedInteger32	one of: 0, 1, 2,
sensorOffsets		Array	
–>length	4	UnsignedInteger32	<= 2, >= 0
–>elements	8	UnsignedInteger64	<= 20000000
hasIntensities	1	Boolean	No restrictions
minIntensity	4	IEEE754Single	No restrictions
maxIntensity	4	IEEE754Single	No restrictions
intensities		Array	
–>length	4	UnsignedInteger32	<= 20000000, >= 0
–>elements	4	IEEE754Single	No restrictions
optionalData		Extension	
extensionSize	4	UnsignedInteger32	No restrictions
fieldOffset		Array	
–>length	4	UnsignedInteger32	No restrictions
–>elements	4	UnsignedInteger32	No restrictions
Field 0: reflectors		Array	
–>length	4	UnsignedInteger32	<= 20000000 >= 0

Name	Size (B)	Type	Restriction
–>elements	4	PointCluster	0 <= from <= to <= 19999999; |x|,|y| <= 1.e+12

- **timestamp**: The time at which the ROKIT Locator 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**: The estimated Cartesian (x,y)-coordinates (in meters) and yaw angle (in radians) of the vehicle, given in the map reference frame.
- **distanceToLastLC**: Distance traversed by the sensor since the last successful loop closure (in meters).
- **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. The (x,y) coordinates of each point are given in meters.
- **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 ClientRecording-PathTypeEnum.
- **sensorOffsets**: Array of indices of the point cloud field **scan**. The i-th index represents the first scan point contributed by the i-th laser sensor. In interval notation the points that belong to the i-th sensor have indices within the interval [sensorOffsets[i], sensorOffsets[i+1]). If i is the index of the last sensor, it contributes the points with the indices [sensorOffsets[i], **scan**–>length).
- **hasIntensities**: Boolean indicating whether intensities are available. When set to false, the minIntensity and maxIntensity field are undefined and the intensities array has a length of zero.
- **minIntensity**: IEEE754Single indicating the minimum value for the entries in the intensities array.
- **maxIntensity**: IEEE754Single indicating the maximum value for the entries in the intensities array.
- **intensities**: Array of IEEE754Single containing the intensity values for the laser scan. Contains either the same number of elements as the scan, or zero elements.
- **optionalData**: Start of the Extension of this datagram. Provides its own size to enable the user to skip it easily.
- **reflectors**: Array of PointCluster. Each cluster defines points within the **scan** point cloud that were classified as a single reflector marker. Its prototype describes the estimated position (in meters) and normal angle (in radians) of the reflector.

ClientRecordingRecordingStatus

Label	Value (Signed-Integer32)	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 (Signed-Integer32)	Description
NORMAL	0	Path node of the current recording
POSSIBLE_LOOP_CLOSURE	1	Path node that should be revisited to make a loop closure
POSSIBLY_VISIBLE_LOOP_CLOSURE	2	POSSIBLE_LOOP_CLOSURE that lies within the field of view of the most recent laser scan

13.6.2 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 ROKIT Locator Client is in recording mode and uses the ClientRecordingMapDatagram.

ClientRecordingMapDatagram

Name	Size (B)	Type	Restriction
map		Array	
–>length	4	UnsignedInteger32	<= 134217728, >= 0
–>elements	8	Position2DSingle	|x|,|y| <= 1.e+05
optionalData		Extension	
extensionSize	4	UnsignedInteger32	No restrictions
fieldOffset		Array	
–>length	4	UnsignedInteger32	No restrictions
–>elements	4	UnsignedInteger32	No restrictions
Field 0: reflectors		Array	
–>length	4	UnsignedInteger32	<= 134217728, >= 0

Name	Size (B)	Type	Restriction
–>elements	4	PointCluster	0 <= from <= to <= 134217727; |x|,|y| <= 1.e+12

- **map**: A 2D point cloud representing the current state of the map during the recording process. The (x,y) coordinates of each point are given in meters.
- **optionalData**: Start of the Extension of this datagram. Provides its own size to enable the user to skip it easily.
- **reflectors**: Array of PointCluster. Each cluster defines points within the **map** point cloud that were classified as a single reflector marker. Its prototype describes the estimated position (in meters) and normal angle (in radians) of the reflector.

13.7 MAPPING OUTPUT (MODULE: *CLIENTMAP*)

The API Module ClientMap contains the following binary interfaces:

ClientMapVisualization provides information about the current map building process.

ClientMapMap provides the current state of the map.

In the following sections these interfaces are detailed.

13.7.1 ClientMapVisualization Interface

The ClientMapVisualization interface provides information needed to visualize and evaluate the current map building process. It operates while the ROKIT Locator 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.

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
progress	8	IEEE754Double	<= 1.e+00, >= 0.e+00
scan		Array	

Name	Size (B)	Type	Restriction
–>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, yaw in $[-\pi_f, \pi_f]$
pathTypes		Array	
–>length	4	UnsignedInteger32	<= 20000000, >= 0
–>elements	4	SignedInteger32	one of: 0, 1, 2,
sensorOffsets		Array	
–>length	4	UnsignedInteger32	<= 2, >= 0
–>elements	8	UnsignedInteger64	<= 20000000
hasIntensities	1	Boolean	No restrictions
minIntensity	4	IEEE754Single	No restrictions
maxIntensity	4	IEEE754Single	No restrictions
intensities		Array	
–>length	4	UnsignedInteger32	<= 20000000, >= 0
–>elements	4	IEEE754Single	No restrictions
optionalData		Extension	
extensionSize	4	UnsignedInteger32	No restrictions
fieldOffset		Array	
–>length	4	UnsignedInteger32	No restrictions
–>elements	4	UnsignedInteger32	No restrictions
Field 0: reflectors		Array	
–>length	4	UnsignedInteger32	<= 20000000 >= 0
–>elements	4	PointCluster	0 <= from <= to <= 19999999; |x|,|y| <= 1.e+12

- **timestamp**: The time at which the ROKIT Locator 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**: The estimated Cartesian (x,y)-coordinates (in meters) and yaw angle (in radians) of the vehicle, given in the map reference frame.
- **distanceToLastLC**: Distance traversed by the sensor since the last successful loop closure (in meters).
- **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. The (x,y) coordinates of each point are given in meters.
- **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 ClientRecording-PathTypeEnum.
- **sensorOffsets**: Array of indices of the point cloud field **scan**. The i-th index represents the first scan point contributed by the i-th laser sensor. In interval notation the points that belong to the i-th sensor have indices within the interval [sensorOffsets[i], sensorOffsets[i+1]). If i is the index of the last sensor, it contributes the points with the indices [sensorOffsets[i], **scan**–>length).
- **hasIntensities**: Boolean indicating whether intensities are available. When set to false, the minIntensity and maxIntensity field are undefined and the intensities array has a length of zero.
- **minIntensity**: IEEE754Single indicating the minimum value for the entries in the intensities array.
- **maxIntensity**: IEEE754Single indicating the maximum value for the entries in the intensities array.
- **intensities**: Array of IEEE754Single containing the intensity values for the laser scan. Contains either the same number of elements as the scan, or zero elements.
- **optionalData**: Start of the Extension of this datagram. Provides its own size to enable the user to skip it easily.
- **reflectors**: Array of PointCluster. Each cluster defines points within the **scan** point cloud that were classified as a single reflector marker. Its prototype describes the estimated position (in meters) and normal angle (in radians) of the reflector.

ClientMapMappingStatus

Label	Value (Signed-Integer32)	Description
STARTUP	-2	Map building process is starting up and not yet running
DELAYED	1	Unused
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 the currently processed scan

13.7.2 ClientMapMap Interface

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

ClientMapMapDatagram

Name	Size (B)	Type	Restriction
map		Array	
–>length	4	UnsignedInteger32	<= 134217728, >= 0
–>elements	8	Position2DSingle	|x|,|y| <= 1.e+05
optionalData		Extension	
extensionSize	4	UnsignedInteger32	No restrictions
fieldOffset		Array	
–>length	4	UnsignedInteger32	No restrictions
–>elements	4	UnsignedInteger32	No restrictions
Field 0: reflectors		Array	
–>length	4	UnsignedInteger32	<= 134217728, >= 0
–>elements	4	PointCluster	0 <= from <= to <= 134217727; |x|,|y| <= 1.e+12

- **map**: A 2D point cloud representing the current state of the map during the mapping process. The (x,y) coordinates of each point are given in meters.
- **optionalData**: Start of the Extension of this datagram. Provides its own size to enable the user to skip it easily.
- **reflectors**: Array of PointCluster. Each cluster defines points within the **map** point cloud that were classified as a single reflector marker. Its prototype describes the estimated position (in meters) and normal angle (in radians) of the reflector.

13.8 GLOBAL ALIGNMENT (MODULE: *CLIENTGLOBALALIGN*)

13.8.1 ClientGlobalAlignVisualization Interface

The ClientGlobalAlignVisualization interface provides information needed to visualize and evaluate global alignment landmarks and their observations. It operates while the ROKIT Locator 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.

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, yaw in $[-\pi_f, \pi_f]$
landmarks		Array	
–>length	4	UnsignedInteger32	<= 20000000, >= 0
–>elements	var.	ClientGlobalAlign-LandmarkVisualization-Information	No restrictions.
observations		Array	
–>length	4	UnsignedInteger32	<= 20000000, >= 0
–>elements	8	ClientGlobalAlign-LandmarkVisualization-Notice	No restrictions.

- **timestamp**: The time at which the ROKIT Locator 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 ROKIT Locator, specified through ClientGlobalAlignLandmarkVisualizationInformation.
- **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
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, yaw in $[-\pi_f, \pi_f]$
type	8	SignedInteger64	May only contain values defined within the ClientGlobalAlignObservationType.
hasOrientation	1	Boolean	No restrictions.
name		Array	
–>length	4	UnsignedInteger32	<= 240, >= 0
–>elements	1	PrintableAsciiCharacter	No restrictions.

- **pose**: Pose2DSingle encoding the pose of the landmark as estimated by the ROKIT Locator.
- **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.

13.9 SENSOR INPUT (MODULE: *CLIENTSENSOR*)

The API Module ClientSensor contains the following binary interfaces:

ClientSensorLaser offers an alternative way to provide laser measurement data to the ROKIT Locator.

ClientSensorLaserOutput provides low-latency output of the laser measurement data.

ClientSensorOdometry provide odometry data to the ROKIT Locator.

ClientSensorInertialMeasurementUnit provide IMU data to the ROKIT Locator.

ClientSensorAbsolutePose provide absolute pose data to the ROKIT Locator.

In the following sections these interfaces are detailed.

13.9.1 ClientSensorLaser Interface

The ClientSensorLaser interface offers an alternative to the ROKIT Locator 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 ROKIT Locator Client then uses these laser scans instead of the scans acquired through a built-in driver. To use this interface, set the client's `ClientSensor.laser.type` 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 `ClientSensor.laser.address` to the address and port of the external laser source. Once connected, this interface expects the user to send ClientSensorLaserDatagrams. Since the ROKIT Locator is running in docker, the `ClientSensor.laser.address` 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.

If the ClientSensorLaser interface is used without transmitting intensity values for the laser scans, then laser intensities have to be deactivated via the `ClientSensor.laser.useIntensities` configuration parameter.

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 ROKIT Locator.

The ClientSensorLaser interface has to adapt some behaviors originating 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 provide 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.

ClientSensorLaserDatagram This diagram describes a laser scan captured by a two-dimensional rotating laser scanner. The ROKIT Locator 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 ROKIT Locator 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

Name	Size (B)	Type	Restriction
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
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 (in seconds) that passes between two subsequent beams within a scan.
- **duration_scan**: Time (in seconds) required for a single scan. Equal to the beam duration multiplied with the number of beams.
- **duration_rotate**: Time (in seconds) required 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 (in radians).
- **angleEnd**: Bearing of the last beam within the scan (in radians).
- **angleInc**: Angle between two subsequent beams (in radians). 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 (in meters) which can be measured by the scanner. Measurements below this value will be treated as erroneous and discarded.
- **maxRange**: Highest possible range (in meters) which can be measured by the scanner as specified by the manufacturer. Measurements above this value will be treated as erroneous and discarded.
- **ranges**: Array of ranges (in meters) as measured by the sscanner, each element corresponding to one beam. Thus, the first element within this array corresponds to the range measured by the first beam 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 **ClientLaserMask.maxRange** 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.

13.9.2 ClientSensorLaserOutput Interface

The ClientSensorLaserOutput interface is a push provider that outputs the laser scans which the ROKIT Locator Client receives from the first laser. If dual laser operation is enabled using the configuration, the ClientSensorLaser2Output interface provides laser scans received from the second laser in the same manner. Similarly, the data received from the auxiliary laser scanners 3–6 will bre provided through the ClientSensorLaser3Output–ClientSensorLaser6Output interfaces.

Laser scans are only sent through this interface while the ROKIT Locator Client is connected to the laser scanner(s) and is receiving laser data. This is always the case if the LOC, REC, or VISUALRECORDING client control modes are in the RUN state. The ROKIT Locator Client can also be forced to retrieve and output laser data by setting the LASEROUTPUT client control mode to RUN. This ensures that laser data is retrieved even when the LOC, REC, or VISUALRECORDING client control modes are not used.

The ClientSensorLaserOutput interfaces work in the same manner no matter if laser scans are received using one of the internal laser driver or through the ClientSensorLaser interface. The scans are sent as ClientSensorLaserDatagrams, in the same manner as used by the ClientSensorLaser interface.

Laser scans are provided as received by the ROKIT Locator Client without any additional processing. This ensures that any delay in providing the laser scans is minimized. However, as no preprocessing is performed, the exact nature of the scan data can vary depending on the model and manufacturer of the laser sensor(s) being used. In particular, the laser intensity values contained within the scans can vary significantly for different laser types.

13.9.3 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 ROKIT Locator Client. The Binary Interface will provide this data to the ROKIT Locator 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 vehicle 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 vehicle 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 radians.

The interface is only read when the ROKIT Locator Client receives laser information (from the internal driver or the ClientSensorLaser interface).

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 vehicle 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 sensor 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 (in meters) and yaw angle (in radians) that specify the sensor'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 (in meters per second) and angular velocity in yaw (in radians per second) . As measured at the time **timestamp** and given in the odometry sensor 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.

13.9.4 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 ROKIT Locator 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 vehicle.

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

The interface is only read when the ROKIT Locator Client receives laser information (from the internal driver or the ClientSensorLaser interface).

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 vehicle 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.

Name	Size (B)	Type	Restriction
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 (in radians per second) measured by the IMU around the three axes at the time **timestamp**.
- **a_x**, **a_y**, **a_z**: The linear acceleration (in meters per second per second) 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.

13.9.5 ClientSensorAbsolutePose Interface (beta version)

The ClientSensorAbsolutePose interface is a consumer that connects to an external source of absolute position or pose data. This source must provide absolute position or pose information from a stand-alone localization system that does not depend on the ROKIT Locator Client. Possible sources include Global Navigation Satellite Systems (GNSS, such as GPS), a vehicle tracking solution based on external sensors (such as motion capturing cameras), or a system that detects markers within the environment (such as QR codes). The Binary Interface will

provide this data to the ROKIT Locator Client. Once connected, this interface expects the user to send ClientSensorAbsolutePoseDatagrams. The user can enable this interface through the appropriate configuration entry. They can also set the address of the absolute pose source and toggle the use of TLS encryption this way.

The user must also specify the transformation between the absolute pose reference frame and the position of the vehicle reference frame using the proper configuration entries.

This requires that the coordinates of the sensor are reported in meters. E.g. sending latitude and longitude in degrees will not work and conversion before sending it to the ROKIT Locator Client is within the user's responsibility.

This interface is considered to be a beta feature. Enabling the interface and sending data might have unexpected effects in some situations or configuration settings.

To maintain the connection to the ROKIT Locator Client, at least one ClientSensorAbsolutePoseDatagram should be sent during every one-second interval. If the external sensor does not currently provide any measurements that could be sent, the connection should be kept alive by sending a heartbeat. To do so, send a datagram with a ClientSensorAbsolutePoseType set to `IGNORE`.

The interface is only read when the ROKIT Locator Client receives laser information (from the internal driver or the ClientSensorLaser interface).

ClientSensorAbsolutePoseDatagram This datagram describes a pose or position estimate derived via an external absolute pose localization source. To specify the transformation from the absolute pose sensor frame to the vehicle frame, the user must set the corresponding configuration entry.

Name	Size (B)	Type	Restriction
timestamp	8	IEEE754Double	<= 1.e+12, >= 0.e+00
poseNumber	4	UnsignedInteger32	No restrictions.
sensorType	4	UnsignedInteger32	A ClientSensorAbsolutePoseType.
x	8	IEEE754Double	<= 1.e+12, >= -1.e+12
y	8	IEEE754Double	<= 1.e+12, >= -1.e+12
yaw	8	IEEE754Double	in $[-\pi_d, \pi_d]$
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

- **timestamp**: The time when this pose estimate of the sensor 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.
- **poseNumber**: Must be incremented between each subsequent datagram being sent.
- **sensorType**: Specifies the type of sensor from which the absolute pose or position information was determined. Also specifies whether the sensor provides a pose or a position. A pose includes orientation information (yaw). If the sensor provides only a position, the yaw value included in this datagram is ignored.
- **x**, **y**, **yaw**: The Cartesian (x,y) coordinates (in meters) and yaw angle (in radians) that specify the sensor's absolute pose in the map at the time given by **timestamp**.
- **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 sensor. 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. For a position sensor, only the entries **1,1**, **1,2**, and **2,2** are used.

ClientSensorAbsolutePoseType Note that this feature is still in beta. The choice of ClientSensorAbsolutePoseType does not have any effect yet, except for the IGNORE type. Currently all sensor types, except for IGNORE, are expected to provide a full pose.

For now, please always use GENERIC_POSE to maintain compatibility with future versions which might handle the other types differently.

Label	Value (UnsignedInteger32)	Description
GENERIC_POSITION	0x00000000 (0)	Generic sensor that provides position data
GENERIC_POSE	0x00000001 (1)	Generic sensor that provides pose data
LANDMARK_POSITION	0x00010000 (65536)	Landmark-tracking sensor (i.e., QR code camera) that provides position data
LANDMARK_POSE	0x00010001 (65537)	Landmark-tracking sensor (i.e., QR code camera) that provides pose data
GNSS_POSITION	0x00020000 (131072)	Satellite navigation sensor (i.e., GPS) that provides position data
GNSS_POSE	0x00020001 (131073)	Satellite navigation sensor (i.e., GPS) that provides pose data

Label	Value (Unsigned-Integer32)	Description
TRACKER_POSITION	0x00030000 (196608)	External tracking sensor (i.e., motion capturing) that provides position data
TRACKER_POSE	0x00030001 (196609)	External tracking sensor (i.e., motion capturing) that provides pose data
IGNORE	0xffff0000 (4294901760)	All data with this type is ignored (use this type as a sensor heartbeat only)

13.10 MAP EXPANSION OUTPUT (MODULE: *CLIENTEXPANDMAP*)

The API Module ClientExpandMap contains the following binary interfaces:

ClientExpandMapVisualization published visualization information about the current map expansion process.

ClientExpandMapPriorMap publishes the current prior map.

ClientRecordingMap publishes the current expanded map.

In the following sections these interfaces are detailed.

13.10.1 ClientExpandMapVisualization Interface

The ClientExpandMapVisualization interface provides information needed to visualize and evaluate the current visual recording process for map expansion. It operates while the ROKIT Locator Client is in mode VISUALRECORDING and uses the ClientExpandMapVisualization-Datagram. This datagram contains the most recent laser scan in the map reference frame. It also contains the overwrite areas and possible loop closure candidates in the prior map.

ClientExpandMapVisualizationDatagram

Name	Size (B)	Type	Restriction
timestamp	8	IEEE754Double	<= 1.e+12, >= 0.e+00
visualizationId	8	UnsignedInteger64	No restrictions.
zones		Array	
–>length	4	UnsignedInteger32	<= 20000000, >= 0
–>elements	var	ClientExpandMap-OverwriteZoneInformation	No restrictions.
priorMapPoses		Array	
–>length	4	UnsignedInteger32	<= 20000000, >= 0
–>elements	12	Pose2DSingle	|x|,|y| <= 1.e+05, yaw in $[-\pi_f, \pi_f]$
priorMapPose-Types		Array	

Name	Size (B)	Type	Restriction
–>length	4	UnsignedInteger32	<= 20000000, >= 0
–>elements	4	SignedInteger32	one of: 0, 1, 2,

- **timestamp**: The time at which the ROKIT Locator 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.
- **zones**: Array of overwrite areas. Each zone is given as a ClientExpandMapOverwriteZoneInformation
- **priorMapPoses** : Array of Pose2DSingle relative to the prior map. The meaning of each pose is given by the corresponding entry in the priorMapPoseTypes array.
- **priorMapPoseTypes** : Array of ClientExpandMapPriorMapPoseType.

ClientExpandMapOverwriteZoneInformation Contains information about an overwrite area to be used for visualization.

Name	Size (B)	Type	Restriction
polygon		Array	
–>length	4	UnsignedInteger32	<= 20000000, >= 0
–>elements	8	Position2DSingle	|x|,|y| <= 1.e+05
id	8	SignedInteger64	No restrictions.
type	8	SignedInteger64	May only contain values defined within the ClientExpandMapOverwriteZoneType.
name		Array	
–>length	4	UnsignedInteger32	<= 240, >= 0
–>elements	1	PrintableAsciiCharacter	No restrictions.

- **polygon**: Array of Position2DSingle representing the points of a polygon.
- **id**: SignedInteger64 indicating the unique id of the overwrite area.
- **type**: SignedInteger64 indicating the type of the landmark according to the ClientExpandMapOverwriteZoneType.
- **name**: Array of PrintableAsciiCharacter representing the unique name assigned to the overwrite area.

ClientExpandMapPriorMapPoseType

Label	Value (SignedInteger32)	Description
NORMAL	0	Prior map pose of the current recording
POSSIBLE_LOOP_CLOSURE	1	Prior map pose 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

13.10.2 ClientExpandMapPriorMap Interface

The ClientExpandMapPriorMap provides the prior map which the user wants to expand within the current recording process. It is a static map and uses the ClientExpandMapMapDatagram.

ClientExpandMapMapDatagram

Name	Size (B)	Type	Restriction
map		Array	
–>length	4	UnsignedInteger32	<= 134217728, >= 0
–>elements	8	Position2DSingle	|x|,|y| <= 1.e+05
optionalData		Extension	
extensionSize	4	UnsignedInteger32	No restrictions
fieldOffset		Array	
–>length	4	UnsignedInteger32	No restrictions
–>elements	4	UnsignedInteger32	No restrictions
Field 0: reflectors		Array	
–>length	4	UnsignedInteger32	<= 134217728, >= 0
–>elements	4	PointCluster	0 <= from <= to <= 134217727; |x|,|y| <= 1.e+12

- **map**: A 2D point cloud representing the prior map used in the map expansion. The (x,y) coordinates of each point are given in meters.
- **optionalData**: Start of the Extension of this datagram. Provides its own size to enable the user to skip it easily.
- **reflectors**: Array of PointCluster. Each cluster defines points within the **map** point cloud that were classified as a single reflector marker. Its prototype describes the estimated position (in meters) and normal angle (in radians) of the reflector.

13.10.3 ClientRecordingMap Interface

The ClientRecordingMap of the module ClientRecording is used to provide the current state of the map that could be generated using this recording.

13.11 PORT OVERVIEW

The interfaces provided by the ROKIT Locator use the following ports:

Table 92: Default Port Definition

Interface	Port	Secure Port (TLS)
JSON RPC ROKIT Locator Client	8080	8443
JSON RPC ROKIT Locator Server	8082	8445
JSON RPC ROKIT Locator Client Support	8084	8447
JSON RPC ROKIT Locator Server Support	8086	8449
Binary SystemStateHeartbeat ROKIT Locator Server	9000	9440
Binary SystemStateHeartbeat ROKIT Locator Server Support	9001	9441
Binary SystemStateHeartbeat ROKIT Locator Client	9002	9442
Binary SystemStateHeartbeat ROKIT Locator Client Support	9003	9443
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
Binary ClientExpandMapVisualization	9013	9453
Binary ClientExpandMapPriorMap	9014	9454
Binary DiagnosticEvent ROKIT Locator Server	9015	9455
Binary DiagnosticEvent ROKIT Locator Client	9016	9456
Binary SystemJsonRequestIdList ROKIT Locator Server	9017	9457
Binary SystemJsonRequestIdList ROKIT Locator Server Support	9018	9458
Binary SystemJsonRequestIdList ROKIT Locator Client	9019	9459
Binary SystemJsonRequestIdList ROKIT Locator Client Support	9020	9460
Binary ClientSensorLaserOutput	9021	9461
Binary ClientSensorLaser2Output	9022	9462

Interface	Port	Secure Port (TLS)
Binary DiagnosticStatusMonitoring ROKIT Locator Server	9023	9463
Binary DiagnosticStatusMonitoring ROKIT Locator Client	9024	9464
Binary ClientSensorLaser3Output	9025	9465
Binary ClientSensorLaser4Output	9026	9466
Binary ClientSensorLaser5Output	9027	9467
Binary ClientSensorLaser6Output	9028	9468
Binary Interface Proxy ROKIT Locator Server	9090	-
Binary Interface Proxy ROKIT Locator Client	9091	-
Internal: ROKIT Locator Server Port for ROKIT Locator Client Communication	-	21638
If Dongle licensing is used: Wibu Codemeter Communication	-	22350

13.12 FAQ

How to encode / decode base64 strings or data ? The encoding and decoding can be done directly on the website www.base64encode.org, or the 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;
```

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.

Deserialize Binary Interface datagrams: Best practice For a general introduction how to implement a TCP interface on linux systems, refer to the code examples given in section 16.3.1. In addition, there are some caveats, tips, and best practices any implementation attempt should follow.

- In a freshly connected TCP connection the first data that arrives will always be the start of a new datagram. This can be used as an error recovery mechanism. Whenever the deserialization of a message fails, times out, or yields unspecified results, the suggestion is to reconnect to the interface in question.
- All Binary Interface datagrams consist of packed data structures, members of which are called fields in this documentation. Each field has the specified size and can be deserialized into the specified data type. The TCP protocol ensures that the order of these fields is preserved. Thus, there are no keys, magic numbers, headers, tails, or identifiers that define the meaning of the data, nor are they necessary. Instead, the meaning of each field is uniquely determined by the sizes of the preceding fields and, thus, by its position in the datagram.
- Some datagrams carry extensions. The first field in each extension is the `size` of this extension. Always implement the deserialization process in such a way that the complete extension is read, i.e., in total exactly `size` bytes are read. To skip the whole extension, just read the `size` into an unsigned int with four bytes length, subtract the four bytes already read, and discard this number of bytes. This can be done by reading the required number of bytes into a temporary buffer.
- The ROKIT Locator guarantees output validation of its datagrams. This means that a datagram will never be submitted to the customer, if the data fields do not adhere to the specifications in this document. However, it is encouraged to validate the datagrams again–against the same specification–during or after deserialization. This way any incompatibility, be it due to an implementation error or an API version mismatch, will quickly become apparent.

In addition, there are two major architectural concepts that may be used when communicating with the Binary Interface. Both make it possible to timely read the data, determine the meaning of each field and deserialize the data to the appropriate data type. The following list provides a short discussion on these concepts and tips to avoid implementation errors.

- Event-driven architecture: Whenever the receiving system receives a data transmission from the ROKIT Locator, the operating system will publish an event or signal. This event will trigger a function that handles the data for the appropriate interface from which the data was sent. This function must then be able to determine which field the data belongs to. To do this, each interface must have a journal that contains information whether an incomplete datagram is already present and at which position of this datagram the last received data has been stopped. This journal is paramount to determine which data field must be filled next. This way one can ensure that data fragmented into multiple data transmissions can be handled correctly. Due to the properties of the TCP protocol, the order of the transmissions will always be correct.
- Thread-driven architecture: In this case, each interface is handled by its own thread that continuously waits to receive and read data. Therefore, when receiving a data transmission from the ROKIT Locator, this thread can start or continue handling the data and fill

a datagram. In this–much simpler–case all necessary calls to read data can be made successively and no journaling is necessary since each read call just waits until the necessary data is received. After the whole datagram is received the thread shares the result with the rest of the program and starts reading a completely new datagram.

14 User Groups

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

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

14.1 COMMON API METHODS

The following table shows group-dependent access of common API methods in alphabetic order.

API	admin	user	observer	comment
aboutBuildList	x	x	x	all
aboutBuildVersion	x	x	x	all including not authenticated user
aboutModulesList	x	x	x	all including not authenticated user
aboutModulesComponent	x	x	x	all including not authenticated user
certificateSet	x			admin
configList	x	x	x	all
configDefaultList	x	x	x	all
configSchemaGet	x	x	x	all
configSet	x	x		user, admin
dataExchangeGetDownloadPath	x			admin
dataExchangeGetUploadPath	x			admin
dataExchangeList	x			admin
dataExchangeDelete	x			admin
diagnosticClear	x			admin
diagnosticList	x	x	x	all
internalUserDataGet	x	x	x	all
internalUserDataSet	x	x		user, admin
licensingFeatureGet TrustedPlatformModuleInformation	x	x		user, admin
licensingFeatureGetHostId	x	x		user, admin
licensingFeatureList	x	x	x	all

API	admin	user	observer	comment
licensingFeatureSet	x	x		user, admin
licensingFeatureGetServedLicense	x	x		user, admin
licensingFeatureReturnServedLicense	x	x		user, admin
sessionGroupInfo	x	x	x	all
sessionList	x			admin
sessionLogin	x	x	x	all including not authenticated user
sessionLogout	x	x	x	all
sessionRefresh	x	x	x	all
sessionRevoke	x			admin
sessionRevokeAll	x			admin
sessionRevokeId	x			admin
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
supportRecoveryImport	x			admin
supportRecoveryExport	x			admin
systemShutdown	x			admin
userAdd	x			admin
userChangePassword	x			admin
userChangeOwnPassword	x	x	x	all
userDelete	x			admin
userList	x			admin

API	admin	user	observer	comment
userListGroup	x			admin

14.2 ROKIT LOCATOR CLIENT API METHODS

The following table shows group-dependent access of the ROKIT Locator Client's API methods in alphabetic order.

API	admin	user	observer	comment
clientGlobalAlignAddObservation	x	x		user, admin
clientGlobalAlignDeleteObservations	x	x		user, admin
clientGlobalAlignListObservations	x	x	x	all
clientGlobalAlignReplaceObservations	x	x		user, admin
clientLocalizationDockingModeEnable	x	x		user, admin
clientLocalizationDockingModeDisable	x	x		user, admin
clientLocalizationReset	x	x		user, admin
clientLocalizationSetSeed	x	x		user, admin
clientLocalizationStart	x	x		user, admin
clientLocalizationStop	x	x		user, admin
clientManualAlignGetMap	x	x		user, admin
clientManualAlignSet	x	x		user, admin
clientManualAlignStart	x	x		user, admin
clientManualAlignStop	x	x		user, admin
clientMapDelete	x	x		user, admin
clientMapGetInfo	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

API	admin	user	observer	comment
clientRecordingStopVisualRecording	x	x		user, admin
clientRecordingList	x	x	x	all
clientRecordingRename	x	x		user, admin
clientRecordingSetCurrentPose	x	x		user, admin
clientRecordingExport	x			admin
clientRecordingImport	x			admin
clientSensorGetLaserScan	x	x		user, admin
clientSensorLaserOutputStart	x	x		user, admin
clientSensorLaserOutputStop	x	x		user, admin
clientSensorCalibrateDualLaser	x	x		user, admin
clientSensorCalibrateOdometry	x	x		user, admin
clientUserAdd	x			admin
clientUserChangePassword	x			admin
clientUserChangeOwnPassword	x	x	x	all
clientUserDelete	x			admin
clientUserList	x			admin
clientUserListGroup	x			admin
clientExpandMapEnable	x	x		user, admin
clientExpandMapDisable	x	x		user, admin
clientExpandMapAddOverwriteZone	x	x		user, admin
clientExpandMapDeleteOverwriteZones	x	x		user, admin
clientExpandMap ReplaceOverwriteZones	x	x		user, admin
clientExpandMapListOverwriteZones	x	x		user, admin
clientExpandMapGetPriorMap	x	x		user, admin

14.3 ROKIT LOCATOR SERVER API METHODS

The following table shows group-dependent access of the ROKIT Locator Server's API methods in alphabetic order.

API	admin	user	observer	comment
serverMapAddZone	x	x		user, admin
serverMapDelete	x	x		user, admin

API	admin	user	observer	comment
serverMapDeleteZone	x	x		user, admin
serverMapGetImage	x	x	x	all
serverMapGetImageWithResolution	x	x	x	all
serverMapGetInfo	x	x	x	all
serverMapGetMapThumbnail	x	x	x	all
serverMapGetMap	x	x	x	all
serverMapList	x	x	x	all
serverMapRename	x	x		user, admin
serverMapSetZoneName	x	x		user, admin
serverMapSetZonePolygon	x	x		user, admin
serverMapSetZoneType	x	x		user, admin
serverMapExport	x			admin
serverMapImport	x			admin
serverPostAlignMap	x	x		user, admin
serverPostAlignMapCompressed	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		user, admin
serverInternalFleetConfigGet	x	x	x	all

14.4 ROKIT LOCATOR SERVER SUPPORT API METHODS

The following table shows group-dependent access of the ROKIT Locator Server Support's API methods in alphabetic order.

API	admin	user	observer	comment
supportFleetinfoCreate	x	x		user, admin
supportFleetinfoDelete	x	x		user, admin
supportFleetinfoGetPath	x	x		user, admin

API	admin	user	observer	comment
supportFleetinfoList	x	x	x	all
supportFleetinfoSetDescription	x	x		user, admin

15 Licensing

This chapter provides a list of available methods for host machine identification and specifies for which API methods which license features are required. The user can only use the API methods for which his license is activated.

15.1 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.
IDFILE	Uses a signed licensing ID file for identification.
LOCALSERVER	Connect to the Revenera License Server to request a license.
CTRLX	Use the ctrlX licensing service.

When the ROKIT Locator is running as an app on a ctrlX based operating system, only the licensing method CTRLX is accepted. The API configSet will return WARNING if other licensing methods are requested.

15.2 COMMON API METHODS

No license required. All common API methods can be used without any restrictions.

15.3 ROKIT LOCATOR CLIENT API METHODS

Method name	Required feature license
clientGlobalAlignAddObservation	None (always allowed)
clientGlobalAlignDeleteObservations	None (always allowed)
clientGlobalAlignListObservations	None (always allowed)
clientGlobalAlignReplaceObservations	None (always allowed)
clientLocalizationDockingModeEnable	None (always allowed)
clientLocalizationDockingModeDisable	None (always allowed)
clientLocalizationReset	None (always allowed)
clientLocalizationSetSeed	None (always allowed)
clientLocalizationStart	Feature ”Localization”
clientLocalizationStop	None (always allowed)
clientManualAlignGetMap	None (always allowed)

Method name	Required feature license
clientManualAlignSet	None (always allowed)
clientManualAlignStart	None (always allowed)
clientManualAlignStop	None (always allowed)
clientMapDelete	None (always allowed)
clientMapGetInfo	None (always allowed)
clientMapList	None (always allowed)
clientMapRename	None (always allowed)
clientMapSend	Feature ”Map Creation”
clientMapStart	Feature ”Map Creation”, ”Global Alignment” if needed
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)
clientSensorGetLaserScan	None (always allowed)
clientSensorLaserOutputStart	None (always allowed)
clientSensorLaserOutputStop	None (always allowed)
clientSensorCalibrateDualLaser	None (always allowed)
clientSensorCalibrateOdometry	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)
clientExpandMapEnable	Feature ”Map Expansion”

Method name	Required feature license
clientExpandMapDisable	Feature ”Map Expansion”
clientExpandMapAddOverwriteZone	Feature ”Map Expansion”
clientExpandMapDeleteOverwriteZones	Feature ”Map Expansion”
clientExpandMapReplaceOverwriteZones	Feature ”Map Expansion”
clientExpandMapListOverwriteZones	Feature ”Map Expansion”
clientExpandMapGetPriorMap	Feature ”Map Expansion”

15.4 ROKIT LOCATOR SERVER API METHODS

No license required. All ROKIT Locator Server API methods can be used without any restrictions.

15.5 ROKIT LOCATOR SERVER SUPPORT API METHODS

No license required. All ROKIT Locator Server Support API methods can be used without any restrictions.

15.6 LICENSING AND THE REXROTH ROKIT LOCATOR CLIENT

Operating the ROKIT Locator 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)
LASEROUTPUT	None (always allowed)

15.6.1 Licensing and specific Rexroth ROKIT Locator 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 ROKIT Locator 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.

16 Appendix

16.1 MAPEXPANSION WORKFLOW

To perform a map expansion, the following steps have to be carried out in order:

1. Make sure the ROKIT Locator Client performing the map expansion can reach the ROKIT Locator Server.
2. Assure the ROKIT Locator Client performing the map expansion has the latest version of the map by stopping and starting the localization using clientLocalizationStop and clientLocalizationStart.
3. Move the vehicle within the current map until the ROKIT Locator Client is localized and a good startpose for the map expansion is reached. The start pose for a map expansion should be well within the prior map, not directly at its edge.
4. Stop moving the vehicle, note down the current ROKIT Locator pose and do not move the vehicle thereafter until the noted pose was send to the ROKIT Locator Client (see following steps).
5. Stop the localization using clientLocalizationStop.
6. Enable the map expansion mode using clientExpandMapEnable.
7. Start the recording or visual recording using clientRecordingStart or clientRecordingStartVisualRecording.
8. Set the current pose of the recording to the pose noted down previously using clientRecordingSetCurrentPose.
9. The vehicle may now be moved again and the map expansion recording drive can be carried out.
   - Optionally define zones within the prior map which shall be remapped using clientExpandMapAddOverwriteZone. Information from the prior map will be removed within these zones before the new information from the map expansion is applied.
   - Optionally, in case the prior map was globally reference aligned, you may add landmark observations during the map expansion recording the same way as during the initial recording using clientGlobalAlignAddObservation.
10. Stop the recording or visual recording using clientRecordingStop or clientRecordingStopVisualRecording.
11. Disable the map expansion mode using clientExpandMapDisable.
12. Create a new (expanded) map from the recording using the name of the map expansion recording via clientMapStart.
13. Check if DiagnosticList contains any related warnings.
14. Send the resulting map from the ROKIT Locator Client to the ROKIT Locator Server using clientMapSend.
15. Acquire an image of the map using serverMapGetImage and inspect it for plausibility (visually).
16. Now you may set any ROKIT Locator Client's active map to the expanded map using the ClientLocalization.activeMapName configuration parameter.

16.2 BITFIELD

A bitfield is a data structure with a specified number of bits. In this data structure a single bit or a group of bits can be allocated for specific purposes. In ROKIT Locator the bitfield is usually used as single-bit booleans in Localization Info Flags or as the states of Client Control Mode with groups of 3 bits.

Bitfield Example ClientControlMode Assuming the Binary Interface ClientControlMode receives the datagram with controlMode=300105 after starting the localization. The decimal 300105 has the binary form 1001001010001001001 in which the bits are allocated for the following purposes as defined in ClientControlModeDatagram:

decimal	binary						
	EXP	VREC	MAP	LOC	REC	ALIGN	LASER OUTPUT
300105	001	001	001	**010**	001	001	001
299657	001	001	001	001	**010**	001	001
300169	001	001	001	**010**	**010**	001	001

The 3 bits in each mode represent the ClientState. In the above examples:

- The first example shows that LOC is in RUN state, while all others are in READY.
- The second example shows that REC is in RUN state, while all others are in READY.
- The last example shows that LOC and REC are in RUN state, while all others are in READY.

One possible way to check the states of Client Control Modes is using a bitmask with a bitwise AND-operation. The bitmask can be constructed with the ClientState and the bit position of the mode in ClientControlModeDatagram.

```
unit32_t mode = 300105 # set when reading binary interface
# test whether REC is in READY state
# construct bitmask with READY(1) and REC bitfield position 6
uint32_t bitmask_REC_is_READY = 1 << 6
# use bitwise AND-operator to test the variable mode
uint32_t result = mode & bitmask_REC_is_READY
# result > 0 if REC is in READY state
```

16.3 CODE EXAMPLES

16.3.1 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. Where applicable, they also demonstrate how to connect to Binary Interfaces through the Binary Interface Proxy. 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 with optional proxy support
BinaryInterfaceClientControlModeStruct.h	Definition of a ClientControlModeDatagram

Filename	Function
BinaryInterfaceClientSensorAbsolutePose-Struct.h	Definition of a ClientSensorAbsolutePose-Datagrams
BinaryInterfaceClientSensorLaserStruct.h	Definition of a ClientSensorLaserDatagram
SystemJsonRequestIdListStruct.h	Definition of a SystemJsonRequestIdList-Datagram
BinaryInterfaceDummyLaserData.h	Implementation of a dummy-filled ClientSensorLaserDatagram
BinaryInterfaceLocalizationMapStruct.h	Definition of a ClientLocalizationMapDatagram
BinaryInterfaceLocalizationVisualization-Struct.h	Definition of a ClientLocalizationVisualizationDatagram
BinaryInterfaceMapVisualizationStruct.h	Definition of a ClientMapVisualization-Datagram
BinaryInterfacePoseStruct.h	Definition of a ClientLocalizationPose-Datagram
BinaryInterfaceRecordingVisualizationStruct.h	Definition of a ClientRecordingVisualizationDatagram
SystemStateStruct.h	Definition of a SystemState struct
TestClientControlMode.cpp	Program testing the ClientControlModeInterface
TestClientLocalizationMap.cpp	Program testing the ClientLocalization-MapInterface
TestClientLocalizationPose.cpp	Program testing the ClientLocalizationPoseInterface
TestClientLocalizationVisualization.cpp	Program testing the ClientLocalizationVisualizationInterface
TestClientMapVisualization.cpp	Program testing the ClientMapVisualizationInterface
TestClientRecordingVisualization.cpp	Program testing the ClientRecordingVisualizationInterface
TestClientSensorLaser.cpp	Program testing the ClientSensorLaserInterface
TestClientSensorLaserOutput.cpp	Program testing the ClientSensor-LaserOutputInterface
TestClientSensorAbsolutePose.cpp	Program testing the ClientSensorAbsolutePoseInterface

Filename	Function
TestSystemStateHeartbeat.cpp	Program testing the SystemStateHeartbeat of ROKIT Locator Client, ROKIT Locator Client Support, ROKIT Locator Server, ROKIT Locator Server Support
TestDiagnosticStatusMonitoring.cpp	Program testing the TestDiagnosticStatusMonitoringInterface of ROKIT Locator Client and ROKIT Locator Server
TestSystemJsonRequestIdList.cpp	Program testing the SystemJsonRequestIdListInterface of ROKIT Locator Client and ROKIT Locator Server

16.3.2 JSON-RPC Interface

The following examples replicate a collection of commonly used routines. They demonstrate how certain functions can be achieved or automated. These examples are unspecified frames following the documentation. For further functionality they need to be filled with actual data.

supportReport

```
login :
{
    "id": 0,
    "jsonrpc": "2.0",
    "method": "sessionLogin",
    "params": {
        "query": {
            "timeout": {
                "valid": true,
                "time": 300,
                "resolution": 1
            },
            "password": "",
            "userName": ""
        }
    }
}

supportReportSetDescription :
{
    "id": 0,
    "jsonrpc": "2.0",
    "method": "supportReportSetDescription",
    "params": {
        "query": {
            "sessionId": "",
            "reportDescription": ""
        }
    }
```

```
}

supportReportCreate :
{
    "id": 0,
    "jsonrpc": "2.0",
    "method": "supportReportCreate",
    "params": {
        "query": {
            "sessionId": ""
        }
    }
}

supportReportCreateMinimal :
{
    "id": 0,
    "jsonrpc": "2.0",
    "method": "supportReportCreateMinimal",
    "params": {
        "query": {
            "sessionId": ""
        }
    }
}

supportReportList :
{
    "id": 0,
    "jsonrpc": "2.0",
    "method": "supportReportList",
    "params": {
        "query": {
            "sessionId": ""
        }
    }
}

supportReportGetPath :
{
    "id": 0,
    "jsonrpc": "2.0",
    "method": "supportReportGetPath",
    "params": {
        "query": {
            "sessionId": "",
            "reportName": ""
        }
    }
}
```

```
supportReportDelete :
{
    "id": 0,
    "jsonrpc": "2.0",
    "method": "supportReportDelete",
    "params": {
        "query": {
            "sessionId": "",
            "reportName": ""
        }
    }
}

logout :
{
    "id": 0,
    "jsonrpc": "2.0",
    "method": "sessionLogout",
    "params": {
        "query": {
            "sessionId": ""
        }
    }
}
```

supportRecovery

```
login :
{
    "id": 0,
    "jsonrpc": "2.0",
    "method": "sessionLogin",
    "params": {
        "query": {
            "timeout": {
                "valid": true,
                "time": 300,
                "resolution": 1
            },
            "password": "",
            "userName": ""
        }
    }
}

supportRecoveryCreate :
{
    "id": 0,
    "jsonrpc": "2.0",
```

```
        "method": "supportRecoveryCreate",
        "params": {
            "query": {
                "sessionId": ""
            }
        }
    }

    supportRecoveryList :
    {
        "id": 0,
        "jsonrpc": "2.0",
        "method": "supportRecoveryList",
        "params": {
            "query": {
                "sessionId": ""
            }
        }
    }

    supportRecoveryFrom :
    {
        "id": 0,
        "jsonrpc": "2.0",
        "method": "supportRecoveryFrom",
        "params": {
            "query": {
                "sessionId": "",
                "recoveryName": ""
            }
        }
    }

    supportRecoveryFactoryReset :
    {
        "id": 0,
        "jsonrpc": "2.0",
        "method": "supportRecoveryFactoryReset",
        "params": {
            "query": {
                "sessionId": ""
            }
        }
    }

    supportRecoveryDelete :
    {
        "id": 0,
        "jsonrpc": "2.0",
        "method": "supportRecoveryDelete",
```

```
        "params": {
            "query": {
                "sessionId": "",
                "recoveryName": ""
            }
        }
    }

    logout :
    {
        "id": 0,
        "jsonrpc": "2.0",
        "method": "sessionLogout",
        "params": {
            "query": {
                "sessionId": ""
            }
        }
    }
```

diagnostic

```
    login :
    {
        "id": 0,
        "jsonrpc": "2.0",
        "method": "sessionLogin",
        "params": {
            "query": {
                "timeout": {
                    "valid": true,
                    "time": 300,
                    "resolution": 1
                },
                "password": "",
                "userName": ""
            }
        }
    }

    diagnosticList :
    {
        "id": 0,
        "jsonrpc": "2.0",
        "method": "diagnosticList",
        "params": {
            "query": {
                "sessionId": "",
                "filters" : {
```

```
                    "severityFilter" : [2, 3, 4, 6],
                    "groupFilter" : [0, 1, 2, 3, 4],
                    "timeFilter" : [{
                        "timestamp" : {
                            "valid" : true,
                            "time" : 9223372036854775807,
                            "resolution" : 1000000
                        },
                        "timeInterval" : {
                            "valid" : true,
                            "time" : 1000,
                            "resolution" : 1000000
                        }
                    }]
                }
            }
        }
    }

    diagnosticClear :
    {
        "id": 0,
        "jsonrpc": "2.0",
        "method": "diagnosticClear",
        "params": {
            "query": {
                "sessionId": ""
            }
        }
    }

    logout :
    {
        "id": 0,
        "jsonrpc": "2.0",
        "method": "sessionLogout",
        "params": {
            "query": {
                "sessionId": ""
            }
        }
    }
```

17 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.

17.1 VERSION 1.10

17.1.1 Tested Linux distributions

- Ubuntu 22.04 has been added to the list of tested Linux distributions.
- Support for Ubuntu 16.04.02 is discontinued.

17.1.2 Network ports for Client-Server communication

- Communication between the ROKIT Locator Client and ROKIT Locator Server now only requires a single TCP port (21638 by default).

17.1.3 Changes to common types and methods

- Restricted the `time` field of the common Timestamp type depending on the `resolution`. While larger values of `time` were allowed previously, these could result in undefined behavior due to overflow.
- Restricted the `time` field of the common TimeInterval type depending on the `resolution`. While larger values of `time` were allowed previously, these could result in undefined behavior due to overflow.
- Added the new DataExchangeArchiveName type.

17.1.4 Changes to the Binary Interfaces

- Users can now access all ROKIT Locator Client or ROKIT Locator Server TCP Binary Interfaces through a single TCP port using a Binary Interface Proxy.

17.1.5 Changes to the clientUser module

The module is deprecated and will be removed in the next minor release.

17.1.6 Changes to the serverUser module

The module is deprecated and will be removed in the next minor release.

17.1.7 Changes to the ClientApplication module

- Removed the restriction on the parameter `ClientApplication.mapServerPortsStart`
- `ClientApplication.mapServerHttpProxyUrl` has been added
- `ClientApplication.mapServerHttpProxyToken` has been added

17.1.8 Changes to the ClientRecording module

- Added the new clientRecordingExport method.
- Added the new clientRecordingImport method.
- Added the new ClientRecordingNameResponseMessage.
- Added the new ClientRecordingArchiveNameMessage.
- Added the new ClientRecordingArchiveNameResponseMessage.
- Added the new response code `CLIENTRECORDING_ARCHIVE_INVALID`.
- Added the new response code `CLIENTRECORDING_ARCHIVE_WRONG_TYPE`.
- Added the new response code `CLIENTRECORDING_ARCHIVE_INCOMPATIBLE`.
- Added the new response code `CLIENTRECORDING_NOT_ENOUGH_SPACE`.

17.1.9 Changes to the ClientSensor module

- Added a new UDP-based laser driver `keyenceszvudp` for Keyence SZ-V32N devices.
- Added support for auxiliary laser scanners. While these laser scanners are not used for localization of mapping, the user may use the data from these scanners for other purposes.
- Added the new configuration parameters `ClientSensor.enableLaser{3,4,5,6}`.
- Added the new configuration parameters in the groups `ClientSensor.laser{3,4,5,6}`.
- Added the new ClientSensorLaser{3,4,5,6}Output binary interfaces. These interfaces provide low-latency access to the laser scans recorded by the auxiliary lasers.
- The ClientSensorLaserIndexMessage used by the method clientSensorGetLaserScan now additionally accepts laser indices 2, 3, 4 and 5.

17.1.10 Changes to the DataExchange module

- Added the new DataExchange module.
- Added the new dataExchangeGetDownloadPath method.
- Added the new dataExchangeGetUploadPath method.
- Added the new dataExchangeList method.
- Added the new dataExchangeDelete method.
- Added several new module-specific messages.
- Added several new module-specific objects and types.

17.1.11 Changes to the Diagnostic module

- Removed the optional `filters` field from the DiagnosticQueryMessage. This brings this message in line with the query messages from the other API modules.
- Added the new DiagnosticFilterMessage. This message is identical to the previous DiagnosticQueryMessage.
- The diagnosticList method now uses the DiagnosticFilterMessage as its query.
- Using invalid timestamps or time intervals within a DiagnosticAbsoluteTimeFilter or DiagnosticRelativeTimeFilter is no longer an error. Instead, invalid values have special meanings.
- Using the maximum value for a timestamp or time interval within a DiagnosticAbsoluteTimeFilter or DiagnosticRelativeTimeFilter no longer has any special meaning.
- The DiagnosticStatusMonitoring Interface was reworked to contain its information in a json string.

17.1.12 Changes to the Internal module

- Increased the maximum size of the InternalUserDataSettings.

17.1.13 Changes to the Licensing module

- Added `CTRLX` as a possible value for the config parameter `Licensing.licensingMethod`.
- Added the new response code `LICENSING_CTRLX_NOT_FOUND`.

17.1.14 Changes to the ServerInternal module

- The module is deprecated and will be removed in the next minor release.
- Since this deprecated module is an alias for the internal module, all changes made to the internal module also apply to this module.

17.1.15 Changes to the ServerMap module

- The parameter `ServerMap.timeBetweenMapUpdates` must be $\geq$ 60 seconds and $\leq$ 86400 seconds (24 hours).
- Added the new serverMapExport method.
- Added the new serverMapImport method.
- Added the new ServerMapNameResponseMessage.
- Added the new ServerMapArchiveNameMessage.
- Added the new ServerMapArchiveNameResponseMessage.
- Added the new response code `SERVERMAP_ARCHIVE_INVALID`.
- Added the new response code `SERVERMAP_ARCHIVE_WRONG_TYPE`.
- Added the new response code `SERVERMAP_ARCHIVE_INCOMPATIBLE`.

17.1.16 Changes to the Session module

- The timeout in sessionLogin API query SessionLoginMessage is limited to 30days.

17.1.17 Changes to the SupportRecovery module

- Added the new supportRecoveryExport method.
- Added the new supportRecoveryImport method.
- Added the new SupportRecoveryArchiveNameMessage.
- Added the new SupportRecoveryArchiveNameResponseMessage.
- Added the new response code `SUPPORTRECOVERY_ARCHIVE_INVALID`.
- Added the new response code `SUPPORTRECOVERY_ARCHIVE_WRONG_TYPE`.
- Added the new response code `SUPPORTRECOVERY_ARCHIVE_INCOMPATIBLE`.

17.1.18 Changes to the SupportReport module

- Added the new SupportReportTimespanMessage.
- The supportReportCreate method now uses the SupportReportTimespanMessage as its query. Note that this new message is backwards compatible with the existing SupportReportQueryMessage, and thus existing calls to supportReportCreate will continue to work.
- Added the new response code `SUPPORTREPORT_TIMESPAN_INVALID`.

17.2 VERSION 1.9

17.2.1 Changes to the Diagnostic module

- Added the Binary Interface DiagnosticStatusMonitoring accessible via the corresponding port listed in the PortList. DiagnosticStatusMonitoring provides a simple overview about the system status.

17.2.2 Changes to the ClientLocalization module

- Added the new clientLocalizationDockingModeEnable method.
- Added the new clientLocalizationDockingModeDisable method.
- Added the new `CLIENTLOCALIZATION_DOCKING_MODE` localization info flag.
- The parameter ClientLocalization.mapHistoryDuration default value has been changed from 50 seconds to 7500 seconds (> 2 hours).

17.2.3 Changes to the ClientSensor module

- Added a new UDP-based laser driver `sickpicoscanudp` for SICK picoScan devices.
- To avoid numerical issues, the following sensor position configuration parameters are now limited to values between -100 and +100 meters:
  - `ClientSensor.laser.vehicleTransformLaser.{x,y,z}`
  - `ClientSensor.laser2.vehicleTransformLaser.{x,y,z}`
  - `ClientSensor.vehicleErrorModelTransformImu.{x,y,z}`
  - `ClientSensor.vehicleErrorModelTransformOdometry.{x,y,z}`
  - `ClientSensor.vehicleTransformAbsolutePose.{x,y,z}`
  - `ClientSensor.vehicleTransformImu.{x,y,z}`
  - `ClientSensor.vehicleTransformOdometry.{x,y,z}`

17.2.4 Changes to the Config module

- Added the new configSchemaGet method.

17.2.5 Changes to the Licensing module

- Added `IDFILE` as a possible value for the config parameter `Licensing.licensingMethod`.

17.2.6 Changes to the ServerPostAlign module

- Added new module ServerPostAlign.
- Added new method serverPostAlignMap.
- Added new method serverPostAlignMapCompressed.
- Added several new module-specific messages.
- Added several new module-specific objects and types
- Added several new module-specific response codes.

17.2.7 Changes to the User module

- Improved error handling when userAdd receives empty strings as inputs for username or password.

17.2.8 Changes to the ServerMap module

- Added a new zone type, the ServerMapZoneType RESTRICTED_RELOCALIZATION.
- Added a new zone type, the ServerMapZoneType DOCKING.
- Added a new zone type, the ServerMapZoneType DYNAMIC.

17.2.9 Changes to the ClientExpandMap

- Building a map from a map expansion recording will always build the expanded map incorporating the prior map. It is no longer necessary to enable the EXPANDMAP client control mode prior to building a map from a map expansion recording.
- The ClientExpandMapPriorMapName must always be a valid map name, an EmptyString is not allowed anymore.

17.3 VERSION 1.8

17.3.1 Changes to the SupportReport module

- Support reports will now contain at least the latest recording, also if it was recorded longer than two hours ago. If the size of the latest recording is not too large, additional recordings will be added to the report, newest first.

17.3.2 Changes to the aboutBuild module

- Improved the output of the method aboutBuildList.

17.3.3 Changes to common types and methods

- Introduced the Username common JSON object type with a maximum length.
- Introduced the Password common JSON object type with a maximum length.

17.3.4 Changes to the ClientControl module

- The MASK client control mode has been removed and superseeded by the new LASEROUTPUT mode. Use this mode whereever the MASK mode was used before.

17.3.5 Changes to the ClientExpandMap module

- ClientExpandMapOverwriteZoneName has to be between 1 and 240 characters long

17.3.6 Changes to the ClientLaserMask module

- Removed the method clientLaserMaskingGetScan. Use the new method clientSensorGetLaserScan instead.
- Removed the ClientLaserMaskLaserIndexMessage.
- Removed the ClientLaserMaskLaserScanResponseMessage.
- Removed the method clientLaserMaskingStart. Use the new method clientSensorLaserOutputStart instead.
- Removed the method clientLaserMaskingStop. Use the new method clientSensorLaserOutputStop instead.
- Removed the ClientLaserMaskQueryMessage.
- Removed the ClientLaserMaskResponseMessage.

17.3.7 Changes to the ClientLocalization module

- Added the new CLIENTSENSOR_ABSOLUTE_POSE_AVAILABLE localization info flag for use with the ClientLocalizationPoseDatagram.
- Added the new CLIENTSENSOR_ABSOLUTE_POSE_MISSING localization error flag for use with the ClientLocalizationPoseDatagram.
- Added the new `ClientLocalization.allowRelocalization` configuration parameter.

17.3.8 Changes to the ClientSensor module

- Replaced the existing configuration parameter `ClientSensor.sensorFusionMode` with new parameters that support the individual selection of sensors that should be used for localization and mapping:
  - `ClientSensor.sensorFusion.useAbsolutePose`
  - `ClientSensor.sensorFusion.useOdometry`
  - The recording of IMU data is still supported, but the sensor fusion mode with odometry and IMU (IMU_ODO for `ClientSensor.sensorFusionMode`) has been removed.
- Added the new ClientSensorAbsolutePose consumer interface. This interface can be used to send data from an external absolute pose sensor to the ROKIT Locator Client.
- Added the new ClientSensorAbsolutePoseDatagram
- Added the new ClientSensorAbsolutePoseType
- Added new configuration parameters used by the ClientSensorAbsolutePose interface:
  - `ClientSensor.enableAbsolutePose`
  - `ClientSensor.absolutePoseAddress`
  - `ClientSensor.absolutePoseEncryption`
  - `ClientSensor.vehicleTransformAbsolutePose.x`
  - `ClientSensor.vehicleTransformAbsolutePose.y`
  - `ClientSensor.vehicleTransformAbsolutePose.z`
  - `ClientSensor.vehicleTransformAbsolutePose.roll`
  - `ClientSensor.vehicleTransformAbsolutePose.pitch`
  - `ClientSensor.vehicleTransformAbsolutePose.yaw`
- Added the new `ClientSensor.sensorFusion.absolutePoseOverride` configuration parameter. Note: This parameter currently has no effect and is always treated as false.
- Enabled to set the ClientSensor.customReflectorParameters.* individually to -1, whereby the internal default value is used. Thereby not all of the `customReflectorParameters` have to be set when setting `ClientSensor.customReflectorParameters.enabled = true`. Thus, e.g. the internal default laser scanner specific `highIntensityThreshold` can still be used when setting some other values of the `customReflectorParameters` and does not also have to be set manually.
- Added the method clientSensorGetLaserScan. This method replaces the old method clientLaserMaskingGetScan.
- The method clientSensorGetLaserScan requires the ROKIT Locator Client to be in the LASEROUTPUT mode, whereas the deprecated method clientLaserMaskingGetScan required the ROKIT Locator Client to be in the MASK mode.
- Added the new ClientSensorLaserIndexMessage.
- Added the new ClientSensorLaserScanResponseMessage.
- Added the method clientSensorLaserOutputStart used to start the new LASEROUTPUT client control mode.
- Added the method clientSensorLaserOutputStop used to stop the new LASEROUTPUT client control mode.
- Added the new ClientSensorQueryMessage.

- Added the new ClientSensorResponseMessage.
- Removed the upper limit for the elements of the ranges array within the ClientSensorLaserDatagram.
- Added the new ClientSensorLaserOutput and ClientSensorLaser2Output binary interfaces, which provide low-latency access to the laser scans received by the ROKIT Locator Client.
- Added additional ResponseCodes `CLIENTSENSOR_CALIBRATION_NO_VALID_ODOMETRY_DATA` and `CLIENTSENSOR_CALIBRATION_NO_VALID_LASER2_DATA` to the sensor autocalibration to indicate whether sufficient amounts of the required data was contained within the provided recording.

17.3.9 Changes to the Diagnostic module

- Added a new `NOTICE` DiagnosticSeverity level. This level is used for messages describing normal operations. Most existing messages from the `INFO` level have been moved to the `NOTICE` level. The `INFO` level now only contains messages that give regular, timer-based status updates and system usage statistics.

17.3.10 Changes to the Internal module

- This new module is a unified replacement for the now-deprecated ServerInternal module.

17.3.11 Changes to the Licensing module

- The API method licensingFeatureSet is only available if the config parameter `Licensing.licensingMethod` is not set to `LOCALSERVER`.
- The API method licensingFeatureGetServedLicense is only available if the config parameter `Licensing.getServedLicenseAutomatically` is `false` and `Licensing.licensingMethod` is set to `LOCALSERVER`.
- The API method licensingFeatureReturnServedLicense is only available if the config parameter `Licensing.getServedLicenseAutomatically` is `false` and `Licensing.licensingMethod` is set to `LOCALSERVER`.
- Setting the config parameter `Licensing.licensingMethod` to any value other than `LOCALSERVER` will automatically return the served license, if one is available.
- The configuration parameter `Licensing.getServedLicenseAtStartup` has been renamed to `Licensing.getServedLicenseAutomatically`
- New response code `LICENSING_LOCALSERVER_API_NOT_ALLOWED_IN_AUTOMATIC_MODE` added.
- New response code `LICENSING_LOCALSERVER_NO_LICENSE_TO_RETURN` added.
- The response message of the API method licensingFeatureGetServedLicense has been changed to LicensingServedLicenseResponseMessage

17.3.12 Changes to the ServerInternal module

- The ServerInternal module has been deprecated and replaced by the Internal module. The entire module will be removed in the future.
- To maintain compatibility during the transition period, all methods, messages, objects and types from the ServerInternal module are now aliases to their counterparts from the Internal module.

17.3.13 Changes to the ServerMap module

- Added the new configuration parameter `ServerMap.parallelMapDownloads`.
- Server-Client communication requires an additional port. The new port range is 21638-21645 for the default configuration. This is due to a new feature that allows to check server-client compatibility upon connection.
- Added the new response values `currentMapLaserTypes` and `initialMapLaserTypes` to the `serverMapGetInfo` request.
- As part of the introduction of ServerMapZones, the following new API methods have been added:
  - `serverMapAddZone`
  - `serverMapDeleteZone`
  - `serverMapSetZoneName`
  - `serverMapSetZoneType`
  - `serverMapSetZonePolygon`

17.3.14 Changes to the Session module

- The SessionLoginMessage now uses the new Username and Password objects.

17.3.15 Changes to the User module

- The UserAddMessage now uses the common Username and Password objects.
- The UserDeleteMessage now uses the common Username object.
- The UserInfo object now uses the common Username object.
- The UserOwnPasswordChangeMessage now uses the common Password object.
- The UserPasswordChangeMessage now uses the common Username and Password objects.

17.3.16 Changes to the SupportFleetinfo module

- This new module SupportFleetinfo provides APIs for collecting information about the ROKIT Locator Server and all connected ROKIT Locator Clients.
- The port 21646 is additionally required by the new module SupportFleetinfo.

17.4 VERSION 1.7

17.4.1 Changes to the Licensing module

- New API method licensingFeatureGetServedLicense added.
- New API method licensingFeatureReturnServedLicense added.
- New configuration parameter `Licensing.getServedLicenseAtStartup` added.
- New configuration parameter `Licensing.localServerUrl` added.
- New accepted configuration value `LOCALSERVER` for configuration parameter `Licensing.licensingMethod` added.
- New response code `LICENSING_LOCALSERVER_NOT_ALL_FEATURES` added.

17.5 VERSION 1.6

17.5.1 Changes to the Binary Interfaces

- Sending data to a push provider binary interface for which this is not allowed will now result in the connection to the binary interface being closed.

17.5.2 Changes to the ClientApplication module

- Added the new parameter `ClientApplication.mapServerPortsStart`.

17.5.3 Changes to the ClientGlobalAlign module

- The ClientGlobalAlignObservation has been altered: the LASERstaticCalibSENSOR transformation is changed to VEHICLEstaticCalibSENSOR. Instead of providing the transformation between the coordinate system of the first laser and the coordinate system of the sensor that is perceiving the landmark, the customer must now provide the transformation between the coordinate system of the vehicle and the coordinate system of the sensor that is perceiving the landmark.

17.5.4 Changes to the ClientLocalization module

- Clarified that the LOC and VISUALRECORDING client control modes are mutually exclusive.
- Calling clientLocalizationStart while the VISUALRECORDING client control mode is in the RUN state will result in a `NOT_IN_REQUIRED_STATE` error.
- Added the `errorFlags` and `infoFlags` fields to the ClientLocalizationPoseDatagram.
- Diagnostic information was added to the `ClientLocalizationPoseDatagram`. `errorFlags` and `infoFlags` contain diagnostic information on the localization state.

User-written software reading ClientLocalizationPoseDatagram are likely to require adjustment to handle the new data layout.

17.5.5 Changes to the ClientManualAlign module

- clientManualAlignSet will not automatically change the ALIGN client control mode to READY anymore. Use clientManualAlignStop to do so.

17.5.6 Changes to the ClientRecording module

- Clarified that the VISUALRECORDING and LOC client control modes are mutually exclusive.
- Calling clientRecordingStartVisualRecording while the LOC client control mode is in the RUN state will result in a `NOT_IN_REQUIRED_STATE` error.

17.5.7 Changes to the ClientSensor module

- Added methods to automatically calculate the calibration of external sensors from a recording via `clientSensorCalibrateOdometry` and `clientSensorCalibrateDualLaser`.

17.5.8 Changes to the Config module

- configSet will now fail with a `NOW_IN_REQUIRED_STATE` error if a client control mode that uses the configuration parameters is in the RUN state.

17.5.9 Changes to the Diagnostic module

- Added the DiagnosticEvent interface that can be used to receive diagnostic messages as DiagnosticEntryDatagrams. It is accessible via the corresponding port listed in the PortList.

17.5.10 Changes to the ServerMap module

- The map-image coordinate system transformation of the ServerMapPngImageResponseMessage returned by the API method serverMapGetMapThumbnail is now always zero; returning a transformation does not make sense for this method and the transformation returned in the past was incorrect.

17.5.11 Changes to the System module

- A new binary interface, SystemJsonRequestIdList, was added. This interface is available on the ROKIT Locator Client as well as on the ROKIT Locator Server. It is accessible via the corresponding ports listed in the PortList.

17.6 VERSION 1.5

17.6.1 Changes to the Binary Interface

- Introduction of a new field of Binary Interface datagrams called an extension. An extension is a complex binary data type that, like arrays, contains multiple elements of data. In contrast to arrays, however, it encapsulates arbitrary content featuring a header section that allows complete omission of the content as well as deserialization of specific parts of the content. Extensions are only part of a datagram if specified in the datagram's definition and are most typical used to extend complex data types with additional–maybe optional–information.

17.6.2 Changes to common types and methods

- The MapDatagram along with the section Common Datagrams was removed from section 13.1. This change only affects the API modules ClientLocalization, ClientMap, ClientRecording, and ClientExpandMap.
- Introduced the PointCluster common binary type. It contains an index range of points which form a cluster within a given point cloud. Additionally, each cluster is characterized by a binary Pose2DDouble-typed prototype.
- Introduced the PointCluster common JSON object equivalent to the binary PointCluster type. It contains an index range of points which form a cluster within a given point cloud. Additionally, each cluster is characterized by a Pose2D-typed prototype.

17.6.3 Changes to the Certificate module

- The requirements for user certificates changed (refer to the Security chapter of ROKIT Locator 1.10.5 for further information). Old certificates are still supported, but deprecated.

17.6.4 Changes to the Config module

- The configSet RPC method will now report the new `CONFIG_DUPLICATE_PARAMETER_KEY` response code when trying to set a key more than once in a single call.

17.6.5 Changes to the ClientControl module

- Clarified that mode changes may only be requested from the READY to the RUN state or from the RUN to the READY state. Attempting any other mode change will now consistently result in a `NOT_IN_REQUIRED_STATE` response.

17.6.6 Changes to the ClientExpandMap module

- Introduced the ClientExpandMapMapDatagram that replaced the removed common MapDatagram and expanded it by an extension describing reflector markers using the new type PointCluster.
- Renamed the JSON-RPC interface clientExpandMapGetPriorMapPointCloud to clientExpandMapGetPriorMap.
- Expanded ClientExpandMapPriorMapResponseMessage by an array of PointCluster describing reflector markers in the prior map.

17.6.7 Changes to the ClientGlobalAlign module

- Added a new optional entry `timestamp` to the ClientGlobalAlignObservation which can be used to set a specific Timestamp for the observation.

17.6.8 Changes to the ClientLocalization module

- Expanded the ClientLocalizationVisualizationDatagram by an extension describing reflector markers using the new type PointCluster.
- Introduced the ClientLocalizationMapDatagram that replaced the removed common MapDatagram and expanded it by an extension describing reflector markers using the new type PointCluster.
- Introduced the new RPC method clientLocalizationReset that resets all internal states in a running localization. This call, together with the method clientLocalizationSetSeed can be used to recover seamlessly from a major sensor failure or an immediate change of the surroundings without stopping and starting the localization.

17.6.9 Changes to the ClientManualAlign module

- Renamed the JSON-RPC interface clientManualAlignGetPointCloud to clientManualAlignGetMap.
- Renamed ClientManualAlignPointCloud2DResponseMessage to ClientManualAlignMapResponseMessage and expanded the message by an array of PointCluster describing reflector markers in the map.

17.6.10 Changes to the ClientMap module

- Expanded the ClientMapVisualizationDatagram by an extension describing reflector markers using the new type PointCluster.
- Introduced the ClientMapMapDatagram that replaced the removed common MapDatagram and expanded it by an extension describing reflector markers using the new type PointCluster.

17.6.11 Changes to the ClientRecording module

- The ROKIT Locator Client can now detect if the connection to an active laser scanner is lost during a recording or visual recording. In this case, the REC or VISUALRECORDING client control mode will automatically revert from the RUN state to the READY state. The underlying reason will be reported through the diagnostic module. Note that the EXPANDMAP mode remains unchanged by this automatic mode change.
- Expanded the ClientRecordingVisualizationDatagram by an extension describing reflector markers using the new type PointCluster.

- Introduced the ClientRecordingMapDatagram that replaced the removed common MapDatagram and expanded it by an extension describing reflector markers using the new type PointCluster.

17.6.12 Changes to the ClientSensor module

- Clarified that the ports of the `ClientSensor.laser.clientAddress` and the `ClientSensor.laser2.clientAddress` must not be equal when using two lasers with the `sickcola2udp` driver.
- Added new configuration parameters for using reflector markers:
  - `ClientSensor.enableReflectorMarkers`
  - `ClientSensor.customReflectorParameters` (parameter group)

17.6.13 Changes to the ClientUser module

- The ClientUser module has been deprecated and replaced by the User module. The entire module will be removed in the future.
- To maintain compatibility during the transition period, all methods, messages, objects and types from the ClientUser module are now aliases to their counterparts from the User module.

17.6.14 Changes to the ServerMap module

- Renamed the JSON-RPC interface serverMapGetPointCloud to serverMapGetMap.
- Renamed ServerMapPointCloud2DResponseMessage to ServerMapMapResponseMessage and expanded the message by an array of PointCluster describing reflector markers in the map.

17.6.15 Changes to the ServerUser module

- The ServerUser module has been deprecated and replaced by the User module. The entire module will be removed in the future.
- To maintain compatibility during the transition period, all methods, messages, objects and types from the ServerUser module are now aliases to their counterparts from the User module.

17.6.16 Changes to the Session module

- Clarified that the `revokeId` field of the SessionRevokeMessage may not be an empty string.

17.6.17 Changes to the User module

- This new module is a unified replacement for the now-deprecated ClientUser and ServerUser modules.

17.7 VERSION 1.4

17.7.1 Active Sessions Limitations

Changes the inability to log in when the maximum number of Session IDs is reached. After a short-term session has expired the user is now always able to log in with a new short-term session. A short-term session is a session with an expiration timeout less than or equal to 60 seconds.

17.7.2 Laser Intensities

The functionality to read a laser scan's intensity values has been added. Most laser scanners provide intensity information which can be used to improve localization capabilities in the future. Reading these values will be enabled by default. However, a SICK TiM laser scanner is not able to provide this information. Thus, if a SICK TiM laser scanner is used (where this model is only supported for testing, not for regular operational use), laser intensities have to be deactivated via the `ClientSensor.laser.useIntensities` parameter using the API or using the Laser Settings in the aXessor. Otherwise the localization will not work.

17.7.3 Optional entries in JSON-RPC query messages

JSON-RPC query messages may now contain optional entries. Such entries are not mandatory and may be omitted by the user when sending such a message. In this case, the message will be interpreted by the ROKIT Locator as if the missing entry had been set to its default value. The default value for each optional entry is included within the specification of the message to which it belongs. Default values are chosen so that the addition of the new entry does not change the effect of the message.

17.7.4 Changes to the AboutModules module

- Rename the API module identifier of ModuleCertificate to `Certificate` in the ResponseCodeModuleIdentifiers to match the API modules.
- Added new API method aboutModulesComponent to the module AboutModules that queries the component type.

17.7.5 Changes to the Config module

- Added new API method configDefaultList to the module Config that queries a list of all configurable parameters with their default values.
- The parameter ClientLocalization.storeMapHistory default value is set to true.

17.7.6 Changes to the ClientLaserMask module

- When calling clientLaserMaskingGetScan, the user may now specify the index of the scanner for which the scan should be retrieved. This value is optional. If no index is specified, the first laser will be used, preserving the behavior of previous versions.

17.7.7 Changes to the ClientLocalization module

- Calling the clientLocalizationSetSeed method while the ROKIT Locator Client is not self-localizing will now result in an error.
- Added a new optional entry `uncertainSeed` to the ClientLocalizationSeedMessage used by the clientLocalizationSetSeed method.
- The `enforceSeed` entry within the ClientLocalizationSeedMessage now has a slightly different effect when calling the clientLocalizationSetSeed method.

17.7.8 Changes to the ClientSensor module

- In addition to the primary laser, new feature Dual Laser is available supporting a secondary laser.
- For new feature Dual Laser the parameters for the primary laser have been renamed:
  - `ClientLaserMask.maxRange` has been renamed to `ClientLaserMask.laser.maxRange`
  - `ClientLaserMask.minRange` has been renamed to `ClientLaserMask.laser.minRange`
  - `ClientLaserMask.maxRangeLines` has been renamed to `ClientLaserMask.laser.maxRangeLines`
  - `ClientLaserMask.minRangeLines` has been renamed to `ClientLaserMask.laser.minRangeLines`
  - `ClientSensor.laserType` has been renamed to `ClientSensor.laser.type`
  - `ClientSensor.laserAddress` has been renamed to `ClientSensor.laser.address`
  - `ClientSensor.laserClientAddress` has been renamed to `ClientSensor.laser.clientAddress`
  - `ClientSensor.mirrorLaserScans` has been renamed to `ClientSensor.laser.mirrorLaserScans`
- For new feature Dual Laser the parameters for the secondary laser have been added:
  - `ClientSensor.enableLaser2` has been added
  - `ClientLaserMask.laser2.maxRange` has been added
  - `ClientLaserMask.laser2.minRange` has been added
  - `ClientLaserMask.laser2.maxRangeLines` has been added
  - `ClientLaserMask.laser2.minRangeLines` has been added
  - `ClientSensor.laser2.type` has been added
  - `ClientSensor.laser2.address` has been added
  - `ClientSensor.laser2.clientAddress` has been added
  - `ClientSensor.laser2.mirrorLaserScans` has been added
  - `ClientSensor.laser.useIntensities` has been added
  - `ClientSensor.laser2.useIntensities` has been added
- Intensities are now supported and the following configuration entries have been added:
  - `ClientSensor.laser.laserUseIntensities`
  - `ClientSensor.laser2.laserUseIntensities`
- A vehicle coordinate frame has been introduced and the following configuration entries have been added:
  - `ClientSensor.laser.vehicleTransformLaser.x`
  - `ClientSensor.laser.vehicleTransformLaser.y`
  - `ClientSensor.laser.vehicleTransformLaser.z`
  - `ClientSensor.laser.vehicleTransformLaser.roll`
  - `ClientSensor.laser.vehicleTransformLaser.pitch`
  - `ClientSensor.laser.vehicleTransformLaser.yaw`
  - `ClientSensor.laser2.vehicleTransformLaser.x`
  - `ClientSensor.laser2.vehicleTransformLaser.y`
  - `ClientSensor.laser2.vehicleTransformLaser.z`
  - `ClientSensor.laser2.vehicleTransformLaser.roll`
  - `ClientSensor.laser2.vehicleTransformLaser.pitch`
  - `ClientSensor.laser2.vehicleTransformLaser.yaw`
- The parameter group `ClientSensor.laserTransformOdometry` has been renamed to `ClientSensor.vehicleTransformOdometry`
- The parameter group `ClientSensor.laserErrorModelTransformOdometry` has been renamed to `ClientSensor.vehicleErrorModelTransformOdometry`

- The parameter group `ClientSensor.laserTransformImu` has been renamed to `ClientSensor.vehicleTransformImu`
- The parameter group `ClientSensor.laserErrorModelTransformImu` has been renamed to `ClientSensor.vehicleErrorModelTransformImu`

17.7.9 Changes to the Session module

- Add a new API method sessionList which returns a list of current session info objects
- Add a new API method sessionRevoke which revokes a session with a revokation id from an session info object
- Add a new API method sessionRevokeAll which revokes all sessions including the one used to call this method
- Add a new API method sessionRevokeId which returns the revokation id of the calling session

17.7.10 Changes to the SupportRecovery module

- Support recovery point versions 1.1, `1.2` and `1.3` with the known issues.
- The parameter ClientLocalization.storeMapHistory is set to true automatically after applying a recovery point.

17.8 VERSION 1.3

17.8.1 Changes to the Certificate module

- The use of certificate revocation lists can now be disabled using the new parameter `Certificate.enableRevocationList`.

17.8.2 Changes to the ClientApplication module

- New module ClientApplication (contains only config entries) but no methods or message types

17.8.3 Changes to the ClientControl module

- Added new mode EXPANDMAP.

17.8.4 Changes to the ClientExpandMap module

- New module ClientExpandMap.

17.8.5 Changes to the ClientLaserMask module

- Associated config entries have been renamed. Refer to the changelog entry of the Config module for details.

17.8.6 Changes to the ClientLocalization module

- Associated config entries have been renamed. Refer to the changelog entry of the Config module for details.

17.8.7 Changes to the ClientSensor module

- Added new UDP-based laser driver `sickcola2udp` for SICK CoLa2 devices.
- Removed the deprecated `sickmics3` laser driver. Use the `sickcola2` or `sickcola2udp` driver instead.
- Associated config entries have been renamed. Refer to the changelog entry of the Config module for details.

17.8.8 Changes to the Config module

- Added the new parameter `Certificate.enableRevocationList`.
- Added the new parameter `ClientSensor.laserClientAddress`.
- The parameter `LaserComponent.laserType` may now have the new value `sickcola2udp`.
- The parameter `LaserComponent.laserType` may no longer have the value `sickmics3`.
- All existing parameters have been renamed to bring them in line with the API module naming conventions. Each configuration parameter now belongs to a specific API module. The name of each parameter now begins with the name of the module to which it belongs.
- The parameter `application.localization.activeMapName` has been renamed to `ClientLocalization.activeMapName`.
- The parameter `application.localization.autostart` has been renamed to `ClientLocalization.autostart`.
- The parameter `application.localization.mapHistoryDuration` has been renamed to `ClientLocalization.mapHistoryDuration`.
- The parameter `application.mapServerHost` has been renamed to `ClientApplication.mapServerAddress`.
- The parameter `application.vehicleName` has been renamed to `ClientApplication.vehicleName`.
- The parameter `config_laser.mask.max_range` has been renamed to `ClientLaserMask.maxRange`.
- The parameter `config_laser.mask.max_range_line_data` has been renamed to `ClientLaserMask.maxRangeLines`.
- The parameter `config_laser.mask.min_range` has been renamed to `ClientLaserMask.minRange`.
- The parameter `config_laser.mask.min_range_line_data` has been renamed to `ClientLaserMask.minRangeLines`.
- The parameter `config_laser.shall_mirror_scans` has been renamed to `ClientSensor.mirrorLaserScans`.
- The parameter `ExternalSensorComponent.InertialMeasurementUnit.address` has been renamed to `ClientSensor.imuAddress`.
- The parameter `ExternalSensorComponent.InertialMeasurementUnit.enabled` has been renamed to `ClientSensor.enableImu`.
- The parameter `ExternalSensorComponent.InertialMeasurementUnit.tls` has been renamed to `ClientSensor.imuEncryption`.
- The parameter `ExternalSensorComponent.Odometry.address` has been renamed to `ClientSensor.odometryAddress`.
- The parameter `ExternalSensorComponent.Odometry.enabled` has been renamed to `ClientSensor.enableOdometry`.
- The parameter `ExternalSensorComponent.Odometry.tls` has been renamed to `ClientSensor.odometryEncryption`.
- The parameter `global_strategies.laserscan_frontend_imu_odo_mode` has been renamed to `ClientSensor.sensorFusionMode`.

- The parameter group `imu_handler.laser_T_imu` has been renamed to `ClientSensor.laserTransformImu`.
- The parameter `LaserComponent.laserAddress` has been renamed to `ClientSensor.laserAddress`.
- The parameter `LaserComponent.laserType` has been renamed to `ClientSensor.laserType`.
- The parameter `licensing.method` has been renamed to `Licensing.licensingMethod`.
- The parameter `licensing.wibudongle.preferredDongleSerial` has been renamed to `Licensing.preferredDongleSerial`.
- The parameter group `locator_graph.imu_error_model.laser_T_imu` has been renamed to `ClientSensor.laserErrorModelTransformImu`.
- The parameter group `locator_graph.odom_error_model.laser_T_odo` has been renamed to `ClientSensor.laserErrorModelTransformOdometry`.
- The parameter `ltrComponent.enableSupportDataRecorder` has been renamed to `SupportReport.recordDriveData`.
- The parameter `ltrComponent.supportDataDuration_h` has been renamed to `SupportReport.driveDataDurationHours`.
- The parameter `map_update.local_dynamic_map_source.storeMapHistory` has been renamed to `ClientLocalization.storeMapHistory`.
- The parameter `multi_hypotheses_localization.map_update_info.enable` has been renamed to `ClientLocalization.sendMapUpdates`.
- The parameter `mu_server.timeUntilUpdate` has been renamed to `ServerMap.timeBetweenMapUpdates`.
- The parameter group `odometry_handler.laser_T_odo` has been renamed to `ClientSensor.laserTransformOdometry`.
- The parameter `supportrecovery.autosave.enable` has been renamed to `SupportRecovery.enableAutosave`.
- The parameter `supportrecovery.autosave.intervalInDays` has been renamed to `SupportRecovery.autosaveIntervalDays`.
- The parameter `supportrecovery.autosave.numberOfPoints` has been renamed to `SupportRecovery.numberOfAutosaves`.

17.8.9 Changes to the Diagnostic module

- Diagnostic group added to DiagnosticEntry.
- Users not in the admin group may no longer call diagnosticClear.
- Filters added to DiagnosticQueryMessage.
- diagnosticList now supports filters for severity, group and absolute/relative time.

17.8.10 Changes to the Licensing / LicensingFeature module

- The LicensingFeature module has been renamed to the Licensing module, as the naming was previously inconsistent.
- The names of the methods and messages within this module remain unchanged, as they already carried the prefix Licensing previously.

17.8.11 Changes to the ServerInternal module

- Users in the observer group may no longer call serverInternalFleetConfigSet.

17.8.12 Changes to the ServerMap module

- Attempts to call serverMapGetImage with invalid image sizes will now correctly be rejected by the input validation, resulting in a JSON-RPC error. This new behavior is consistent with that of the other JSON-RPC methods.

17.8.13 Changes to the SupportRecovery module

- Associated config entries have been renamed. Refer to the changelog entry of the Config module for details.
- The recovery of the parameter `map_update.local_dynamic_map_source.storeMapHistory` in recovery points 1.2 is not supported. It is a parameter on ROKIT Locator Server in previous version. It is now moved to ROKIT Locator Client and renamed to `ClientLocalization.storeMapHistory` and needs to be set manually.

17.8.14 Changes to the SupportReport module

- Associated config entries have been renamed. Refer to the changelog entry of the Config module for details.

17.8.15 Changes to the System module

- SystemStateHeartbeat added.

17.9 VERSION 1.2.4

While no API changes were made in this patch release with respect to release 1.2 some config entries have changed.

17.9.1 Changes to the Config module

- The following parameter in ConfigEntries has been added:
  - `mu_server.timeUntilUpdate`
- The following parameter in ConfigEntries has been removed:
  - `mu_server.changesUntilUpdate`

17.9.2 Changes to the SupportRecovery module

- Due to the changes in ConfigEntries the recovery of the parameter `mu_server.timeUntilUpdate` from older recovery points with `mu_server.changesUntilUpdate` is not supported. This parameter needs to be set manually.

17.10 VERSION 1.2

The 1.2 release again contains several improvements to the API of the ROKIT Locator. As each of these changes only affects an individual 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.

17.10.1 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.

17.10.2 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.

17.10.3 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 ROKIT Locator Client is not receiving laser sensor data.

17.10.4 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.

17.10.5 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 ROKIT Locator.
- 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 ROKIT Locator 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 within 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 ROKIT Locator. This includes enhanced information on how to encode data from laser scanners with both clockwise and counter-clockwise scanning directions.

17.10.6 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.

17.10.7 Changes to the Config module

- Fixed a bug that could cause the ROKIT Locator 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`

17.10.8 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.

17.10.9 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.

17.10.10 Changes to the ServerMap module

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

17.10.11 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.

17.10.12 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.

17.11 VERSION 1.1

The ROKIT Locator 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.

17.11.1 Network ports for Client-Server communication

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

17.11.2 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.

17.11.3 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

17.11.4 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.

17.11.5 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.

17.11.6 Ensuring coordinate precision

To ensure that floating-point coordinates received and sent by the ROKIT Locator are sufficiently precise, the range of such coordinates has been restricted: Cartesian coordinates that are sent to the ROKIT Locator 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 ROKIT Locator will never generate cartesian coordinates with absolute values exceeding 100,000 meters.

17.11.7 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.

17.11.8 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.

17.11.9 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.

17.11.10 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.

17.11.11 Changes to the Certificate module

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

17.11.12 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.

17.11.13 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 ROKIT Locator Client failed to 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.

17.11.14 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.

17.11.15 Changes to the ClientGlobalAlign module

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

17.11.16 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.

17.11.17 Changes to the ServerMap module

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

17.11.18 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.

17.11.19 Changes to the SupportReport module

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

17.11.20 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.

17.12 VERSION 1.0

The Rexroth ROKIT Locator 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.