

LiquidCan Protocol Documentation

for Distributed Embedded Systems
at the TU Wien Space Team



TU Wien Space Team

Version 0.2

January 5, 2026

Version History

| Version | Date | Changes | Author |
|---------|------------|---|--------------|
| 0.1 | 2025-12-16 | Initial protocol specification proposal | Raffael Rott |
| 0.2 | 2025-12-16 | Implemented Proposals from Team members | Raffael Rott |
| | | | |
| | | | |

Contents

| | |
|---|-----------|
| 1 Purpose and Scope | 4 |
| 2 Notation and Conventions | 4 |
| 2.1 Common Terms | 4 |
| 3 CAN Identifier Scheme & NodeID | 4 |
| 4 Common Frame Layout | 4 |
| 5 Node Registration | 5 |
| 6 Field Registration & Management | 5 |
| 6.1 Registration | 5 |
| 6.2 Regular Operation | 6 |
| 6.2.1 Parameter Updates | 6 |
| 6.2.2 Parameter Setting | 6 |
| 6.2.3 Parameter Locking | 6 |
| 6.2.4 Requesting Field Data | 6 |
| 6.2.5 Field name Lookup | 6 |
| 7 Heartbeats | 7 |
| 8 Status Messages | 7 |
| 9 Message Types | 8 |
| 10 Data Structures | 10 |
| 10.1 NodeInfoRes | 10 |
| 10.2 Status | 10 |
| 10.3 FieldRegistration | 11 |
| 10.4 TelemetryGroupDefinition | 11 |
| 10.5 TelemetryGroupUpdate | 11 |
| 10.6 HeartBeat | 12 |
| 10.7 ParameterSetReq | 12 |
| 10.8 ParameterSetStatus | 12 |
| 10.9 ParameterSetConfirmation | 12 |
| 10.10FieldGetReq | 13 |
| 10.11FieldGetRes | 13 |
| 10.12FieldIDLookupReq | 13 |
| 10.13FieldIDLookupRes | 13 |
| 10.14ParameterSetLock | 14 |
| 10.15DataType | 14 |
| 11 Versioning and Extension Mechanisms | 14 |

1 Purpose and Scope

The purpose of the LiquidCan protocol is to serve at the heart of all future liquids projects at the TU Wien Space Team. Building on the CAN FD standard, it defines how our client devices (such as ECUs) communicate with each other and the central server and with each other. It is designed to be as simple and extensible as possible. One of the design goals of this protocol is to minimize the amount of common type or field definitions between the server and the nodes.

2 Notation and Conventions

- This protocol uses the CAN-FD extension
- All fields are little endian.
- All strings are ASCII encoded and null terminated.
- Payload length: Variable depending on message type but up to 64 Bytes.

2.1 Common Terms

| Term | Description |
|----------------|--|
| (CAN) Client | A CAN client is a device which is connected to the bus |
| Node | Every client which is not the server |
| TelemetryValue | A telemetryValue is a non-externally modifiable value which periodically gets sent to the server |
| Parameters | Parameters can be externally modified |
| Field | An encompassing term for telemetryValues and parameters |
| ECU | A commonly used embedded CAN device at the TU Wien Space Team |

3 CAN Identifier Scheme & NodeID

Each device on the bus has its own unique Node ID. The Server is assigned the Node ID 0. The CAN ID is composed of 11 bits. It contains the sender and receiver Node IDs and a priority bit.

Note the location of the priority bit: It is set as the last bit here since this document expects little-endianness. On the actual bus the priority bit will be sent first, therefore ensuring that the packets are properly prioritised by the CAN Protocol.

| Field | Bits | Description |
|--------------|-----------|----------------------------------|
| Receiver | 5 | Destination node ID |
| Sender | 5 | Source node ID |
| Priority | 1 | Message priority (0=low, 1=high) |
| Total | 11 | Standard CAN ID |

4 Common Frame Layout

The CAN data field consists of 64 bytes. The first byte of each message contains the message type. This simple format allows the protocol to be extended in the future by adding more message types. See Section 9 for a detailed description of message types and their numeric values.

| Field | Bytes | Description |
|--------------|-------|-------------------------------------|
| message_type | 1 | Type of message (see Message Types) |
| data | 63 | Payload data |

5 Node Registration

As soon as a node comes online (or when it receives a `node_info_req`), it enters the registration phase. See Figure 1 for a graphical representation the whole registration procedure. Here it sends out a `node_info_announcement`. This announces the node to the bus and includes its name, number of `telemetryValues` and parameters, and its firmware version through the firmware and LiquidCan hashes (see 10.1). The central server registers the new node and waits for `field_registration` messages. From this point on the node is able to send/receive messages on the bus.

6 Field Registration & Management

Fields are the heart of the protocol. The term Field serves as a general term for both `telemetryValues` and parameters. `TelemetryValues` are periodically sent to the server and non-modifiable. They are meant to represent sensor data or other information which should be periodically logged. Parameters can be externally modified and locked. These are meant for configuration fields, modifiable by either the server or other nodes.

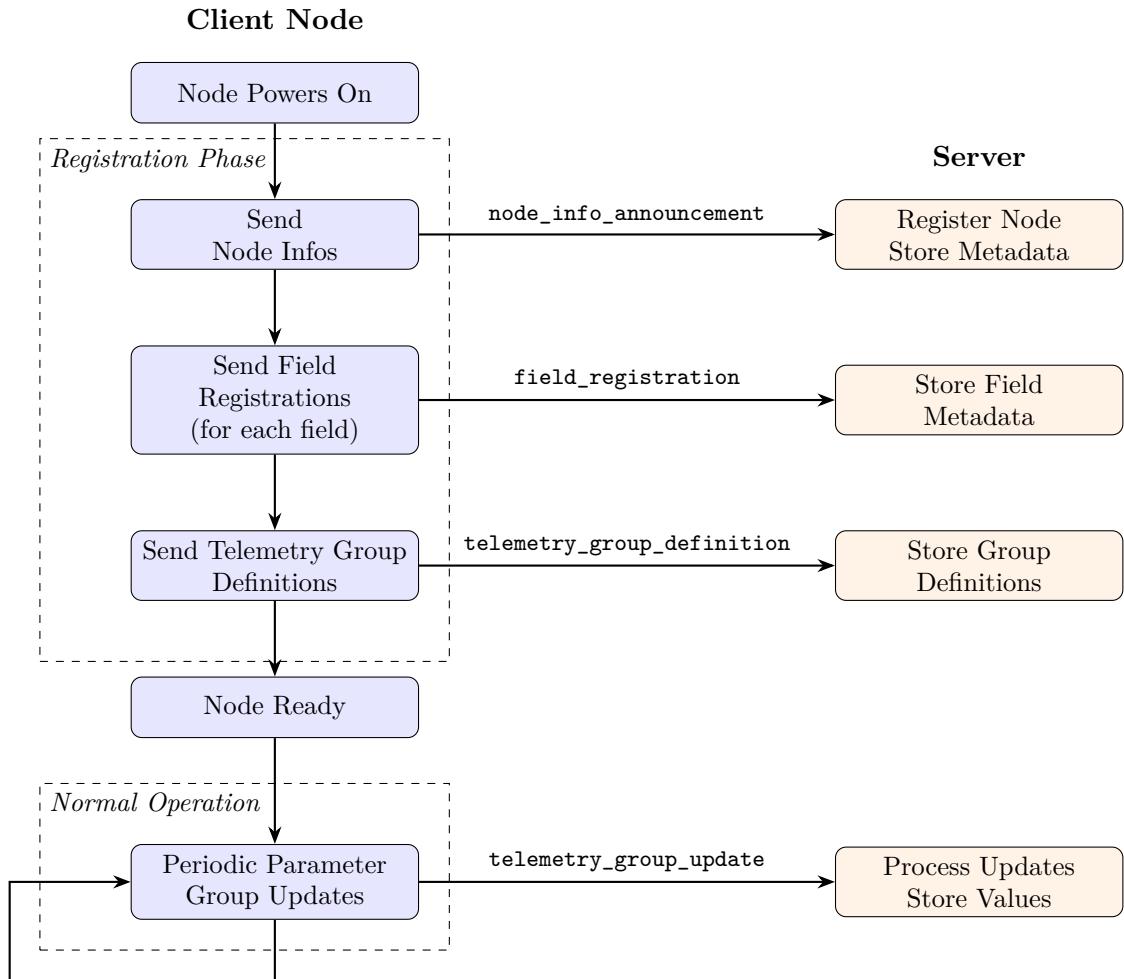


Figure 1: Node Registration and Field Registration Flow

6.1 Registration

`TelemetryValues` and parameters are dynamically defined over the bus.

After registration was initialized, a node sends out `field_registration` messages, one for each parameter/`telemetryValue`. The `FieldRegistration` includes a field ID, the datatype of the field, and a human-readable name. The name and field ID have to be unique per node. The first bit of parameter

field IDs is 0 and the first bit of telemetryValue field IDs is 1. From this point on, the server knows which fields the node has. The fieldID 0 is reserved to indicate an invalid fieldID. It must not be assigned to a field. FieldIDs must be assigned in a deterministic way. This ensures that named fields have the same fieldID between reboots.

Next, the node sends `telemetry_group_definition` messages. The node sends a group ID to identify the group and a list of field IDs. The order of the field IDs must be the same as the order of fields in future `telemetry_group_updates`. The node must ensure that the values of all of the defined `telemetryValues` can fit into a `TelemetryGroupUpdate`.

6.2 Regular Operation

6.2.1 Parameter Updates

During regular operation, each node sends `telemetry_group_update` messages at a defined interval. The order of values here must match the field id ordering of the corresponding `telemetry_group_definition`. The interval can vary between groups, allowing nodes to send `telemetryValues` at different intervals to, for example, reduce bus utilization.

6.2.2 Parameter Setting

Other bus members can send a `parameter_set_req` message, which includes the field ID of the parameter and the new value. Once the node receives the request, it attempts to set the internal parameter value and responds with a `parameter_set_confirmation` message containing a status code, the parameter ID, and the actual value. The status field indicates whether the operation was successful or provides an error code (see `ParameterSetStatus` enum in [10.8](#)). The value field should contain the actual value read back from the parameter, not simply the value that was received in the request.

When a parameter is internally modified through some automated system, the updated value must be sent as a `parameter_set_confirmation` message to the server with a `NodeToNodeModification` status code.

6.2.3 Parameter Locking

A parameter can optionally be locked through a `parameter_set_lock_req` message. After a parameter has been locked, it cannot be modified by an external node. A parameter can only be unlocked by the locking node or the server. To lock a parameter, a node sends a `parameter_set_lock_req` with the fieldID and the locking status (0=unlocked, 1=locked). The receiving node responds with a `parameter_set_lock_confirmation`, confirming the sent fieldID and the locking status. The receiving node also sends a `parameter_set_lock_confirmation` to the server to update it on the locking status of the parameter.

6.2.4 Requesting Field Data

A field can be accessed through a `field_get_req` message, which contains the field ID. Nodes respond with a `field_get_res` message, containing the field ID and the value of the field.

6.2.5 Field name Lookup

The field name lookup covers the case where nodes need to access fields from other nodes. Since they don't receive the `field_registration` messages, they don't know the fieldIDs of the named fields they want to access. `field_id_lookup_req` messages contain the remote field name. The Node responds with a `field_id_lookup_res` message, containing the fieldID and the datatype of the field. A fieldID of 0 indicates that the field was not found.

7 Heartbeats

Heartbeats ensure that the system does not reach a state where it is still dangerous to physically handle but not accessible through CAN messages. The `heartbeat_req` message sent from the server contains a continuously increasing counter. The counter value is unique to each node. If a node does not receive `heartbeat_req` messages, it will default to a safe state. Similarly, the server takes note of any unresponsive nodes.

8 Status Messages

Nodes may send optional status messages to the server. Currently, the three status message types are: `info_status`, `warning_status`, and `error_status`, as defined in section 9. Each message contains a null-terminated string with a status message.

9 Message Types

The following message types are defined in the protocol:

| Enum | Message Type | Data Payload | Direction | Description |
|---------------------------------------|---------------------------------|---------------------------------|--------------------|--|
| <i>Node Discovery and Information</i> | | | | |
| 0 | node_info_req | No payload | Server → Node | Request node information |
| 1 | node_info_announcement | NodeInfoRes (10.1) | Node → Server | Response with node capabilities and identification |
| <i>Status Messages</i> | | | | |
| 10 | info_status | Status (10.2) | Node → Server | Informational status message |
| 11 | warning_status | Status (10.2) | Node → Server | Warning status message |
| 12 | error_status | Status (10.2) | Node → Server | Error status message |
| <i>Field Registration</i> | | | | |
| 20 | telemetryValue_registration | FieldRegistration (10.3) | Node → Server | Register a telemetryValue field |
| 21 | parameter_registration | FieldRegistration (10.3) | Node → Server | Register a parameter field |
| <i>Telemetry Group Management</i> | | | | |
| 30 | telemetry_group_definition | TelemetryGroupDefinition (10.4) | Node → Server | Define a group of fields for batch updates |
| 31 | telemetry_group_update | TelemetryGroupUpdate (10.5) | Node → Server | Update values for a field group |
| <i>Parameter Management</i> | | | | |
| 50 | parameter_set_req | ParameterSetReq (10.7) | Server/Node → Node | Request to set a parameter value |
| 51 | parameter_set_confirmation | ParameterSetConfirmation (10.9) | Node → Server/Node | Response with confirmed parameter value |
| 52 | parameter_set_lock_req | ParameterSetLock (10.14) | Server/Node → Node | Request to lock a parameter |
| 53 | parameter_set_lock_confirmation | ParameterSetLock (10.14) | Node → Server | Response confirming parameter lock |
| <i>Field Access</i> | | | | |
| 60 | field_get_req | FieldGetReq (10.10) | Server/Node → Node | Request field value |
| 61 | field_get_res | FieldGetRes (10.11) | Node → Server/Node | Response with field value |
| 62 | field_id_lookup_req | FieldIDLookupReq (10.12) | Node → Node | Requests the fieldID of a field name |
| 63 | field_id_lookup_res | FieldIDLookupRes (10.13) | Node → Node | Response with field ID |
| <i>Heartbeat</i> | | | | |

| Enum | Message Type | Data Payload | Direction | Description |
|-------------|---------------------|------------------------------------|------------------|--------------------|
| 40 | heartbeat_req | HeartBeat (10.6) | Server → Node | Heartbeat request |
| 41 | heartbeat_res | HeartBeat (10.6) | Node → Server | Heartbeat response |

10 Data Structures

10.1 NodeInfoRes

Response containing information about a node's capabilities.

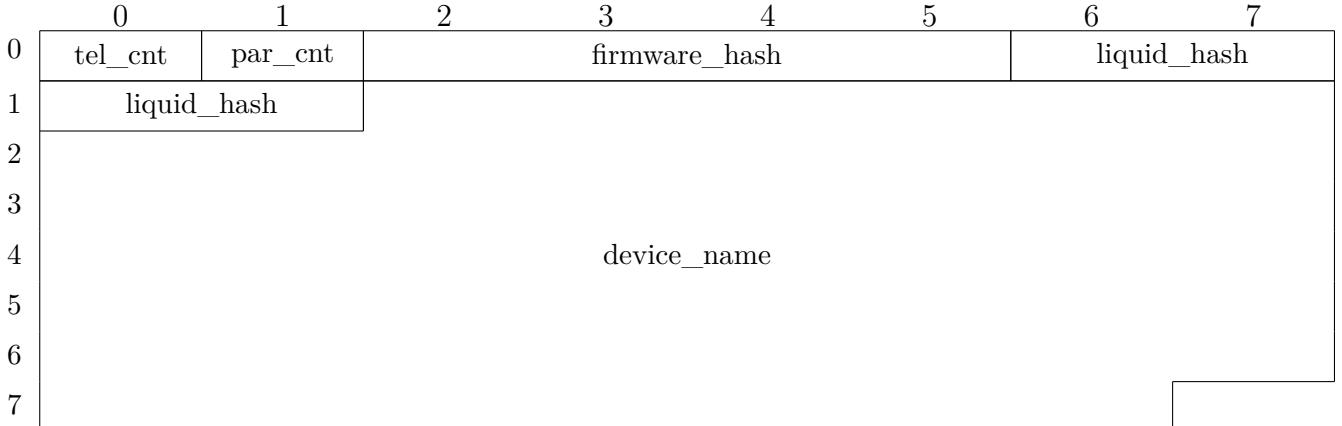


Figure 2: NodeInfoRes byte layout (8 bytes per row)

Field Descriptions:

| Field | Bytes | Description |
|---------------|-----------|--|
| tel_cnt | 1 | Number of telemetryValues on this node |
| par_cnt | 1 | Number of parameters on this node |
| firmware_hash | 4 | Hash of the firmware version |
| liquid_hash | 4 | Hash of the LiquidCan protocol version |
| device_name | 53 | Human-readable device name |
| Total | 63 | |

```
1 typedef struct __attribute__((packed)) {
2     uint8_t var_count;
3     uint8_t par_count;
4     uint32_t firmware_hash;
5     uint32_t liquidk_hash;
6     char device_name[53];
7 } node_info_res_t;
```

Listing 1: NodeInfoRes struct

10.2 Status

General status message with text information.

| Field | Bytes | Description |
|-------|-------|---------------------|
| msg | 63 | Status message text |

```
1 typedef struct __attribute__((packed)) {
2     char msg[63];
3 } status_t;
```

Listing 2: Status struct

10.3 FieldRegistration

Registration information for a telemetryValue or parameter field. The DataType here refers to the DataType Enum value (see [10.15](#)).

| Field | Bytes | Description |
|--------------|-----------|----------------------------------|
| field_id | 1 | Unique identifier for this field |
| field_type | 1 | Data type (DataType enum) |
| field_name | 61 | Human-readable field name |
| Total | 63 | |

```
1 typedef struct __attribute__((packed)) {
2     uint8_t field_id;
3     uint8_t field_type;
4     char field_name[61];
5 } field_registration_t;
```

Listing 3: FieldRegistration struct

10.4 TelemetryGroupDefinition

Defines a group of related telemetryValues for efficient batch updates.

| Field | Bytes | Description |
|--------------|-----------|----------------------------------|
| group_id | 1 | Unique identifier for this group |
| field_ids | 62 | Array of field IDs in this group |
| Total | 63 | |

```
1 typedef struct __attribute__((packed)) {
2     uint8_t group_id;
3     uint8_t field_ids[62];
4 } telemetry_group_definition_t;
```

Listing 4: TelemetryGroupDefinition struct

10.5 TelemetryGroupUpdate

Updates all telemetryValues from a previously defined group.

| Field | Bytes | Description |
|--------------|-----------|--|
| group_id | 1 | Group identifier |
| values | 62 | Packed values of all telemetry values in the group |
| Total | 63 | |

Note: The values are packed in the same order as announced in the TelemetryGroupDefinition.

```
1 typedef struct __attribute__((packed)) {
2     uint8_t group_id;
3     uint8_t values[62];
4 } telemetry_group_update_t;
```

Listing 5: TelemetryGroupUpdate struct

10.6 HeartBeat

Heartbeat message with counter.

| Field | Bytes | Description |
|---------|-------|----------------------------|
| counter | 4 | Incrementing counter value |

```
1 typedef struct __attribute__((packed)) {
2     uint32_t counter;
3 } heartbeat_t;
```

Listing 6: HeartBeat struct

10.7 ParameterSetReq

Request to set a parameter value.

| Field | Bytes | Description |
|--------------|-----------|---------------------------------------|
| parameter_id | 1 | Parameter identifier |
| value | 61 | New value (type depends on parameter) |
| Total | 62 | |

```
1 typedef struct __attribute__((packed)) {
2     uint8_t parameter_id;
3     uint8_t value[61];
4 } parameter_set_req_t;
```

Listing 7: ParameterSetReq struct

10.8 ParameterSetStatus

Status codes for parameter set operations:

| Enum Value | Status Name | Description |
|------------|------------------------|--|
| 0 | Success | Parameter was successfully set |
| 1 | InvalidParameterID | The parameter ID does not exist |
| 2 | ParameterLocked | The parameter is locked and cannot be modified |
| 3 | NodeToNodeModification | The parameter was modified by another node |

10.9 ParameterSetConfirmation

Response to a parameter set request.

| Field | Bytes | Description |
|--------------|-----------|---------------------------------------|
| parameter_id | 1 | Parameter identifier |
| status | 1 | Status code (ParameterSetStatus enum) |
| value | 61 | Confirmed value after set operation |
| Total | 63 | |

```
1 typedef struct __attribute__((packed)) {
2     uint8_t parameter_id;
3     uint8_t status;
4     uint8_t value[61];
5 } parameter_set_confirmation_t;
```

Listing 8: ParameterSetConfirmation struct

10.10 FieldGetReq

Request to retrieve a field value

| Field | Bits/Bytes | Description |
|----------|------------|------------------|
| field_id | 1 Byte | Field identifier |

```
1 typedef struct __attribute__((packed)) {
2     uint8_t field_id;
3 } field_get_req_t;
```

Listing 9: FieldGetReq struct

10.11 FieldGetRes

Response with requested field value.

| Field | Bytes | Description |
|--------------|-------|------------------|
| field_id | 1 | Field identifier |
| value | 62 | Field value |
| Total | | 63 |

```
1 typedef struct __attribute__((packed)) {
2     uint8_t field_id;
3     uint8_t value[62];
4 } field_get_res_t;
```

Listing 10: FieldGetRes struct

10.12 FieldIDLookupReq

Requests the fieldID of a field matching a field name

| Field | Bytes | Description |
|--------------|-------|-------------|
| field_name | 61 | Field Name |
| Total | | 61 |

```
1 typedef struct __attribute__((packed)) {
2     uint8_t field_name[61];
3 } field_id_lookup_req;
```

Listing 11: FieldIDLookupReq struct

10.13 FieldIDLookupRes

Response with requested field id and the datatype of the field.

| Field | Bytes | Description |
|--------------|--------|----------------|
| field_id | 1 Byte | Field ID |
| field_type | 1 Byte | Field Datatype |
| Total | | 63 |

```

1 typedef struct __attribute__((packed)) {
2     uint8_t fieldID;
3     uint8_t field_type;
4 } field_id_lookup_res;

```

Listing 12: FieldIDLookupRes struct

10.14 ParameterSetLock

Locks a parameter to prevent changes.

| Field | Bytes | Description |
|----------------|-------|-----------------------------------|
| parameter_id | 1 | Parameter identifier to lock |
| parameter_lock | 1 | Lock status(0=unlocked, 1=locked) |

```

1 typedef struct __attribute__((packed)) {
2     uint8_t parameter_id;
3     uint8_t lock_status;
4 } parameter_set_lock_t;

```

Listing 13: ParameterSetLock struct

10.15 DataType

The protocol supports the following data types:

| Enum Values | Type Name | Description |
|-------------|-----------|-------------------------|
| 0 | Float32 | 32-bit floating point |
| 1 | Int32 | 32-bit signed integer |
| 2 | Int16 | 16-bit signed integer |
| 3 | Int8 | 8-bit signed integer |
| 4 | UInt32 | 32-bit unsigned integer |
| 5 | UInt16 | 16-bit unsigned integer |
| 6 | UInt8 | 8-bit unsigned integer |
| 7 | Boolean | 1-bit boolean value |

11 Versioning and Extension Mechanisms

This protocol uses semantic versioning. See <https://semver.org/> for a detailed description. A minor update version would, for example, be adding a new datatype or a new message type . Every update of the document must trigger an update of the liquidCAN repo, containing the Rust and C code for each implementation. The updated repo must be reflected in the `liquidcan_hash` field of the `node_info_announcement` used in the firmware/server.

The Firmware in the system can run with minor or patch version differences, "major" changes are not allowed to run simultaneously. The server can inform the operators in case there is firmware running with different major versions.