DI-Net Remote Procedure Protocol (RPC)

Specification

Verification Date:	Validation Date:

Version:	3.3.0	Author(s):	Dual Inventive
Status:	Concept	Date:	07-03-2019

1. General information

1.1. Changelog

Version	Date	Changes
3.3.0	2019-03-06	• error codes: Add DNE_FIRMWARE_BATTERY_TO_LOW when trying to release to empty devices (TWS-74 & TWS-75) • described change impact to RPC-specification for class items with UID (DINET-27) • class monalert: Add create, update, info, subscribe, unsubscribe, list:device, list:user method (BACKEND-4) • class connection: Add info method with connection transports (DINET-39) • removed class dncm (DINET-39) • gps: remove unused siv and fix property from datatype (DINET-40) • error codes: Add DNE_FIRMWARE_LEADER_CONFLICT (DINET-10) • device: Remove gateways method from device class in favor of connection:info (DINET-63) • data type: version is now a structure for multiple versions, hardware and firmware (DINET-57) • class connection: Add nodeid property for can transport (DINET-37) • Add the Reed sensor (BACKEND-150) • class device: device:info: Firmware version property value MAY be a freeform string (DINET-58) • class project: Add project:state method (BACKEND-131) • class connection: connection:info: Explicit document gateway device:uids which are not present (DINET-69) • remove SWITCHBOX 3000 and CRS 3000, these are unused (BACKEND-159) • deprecate dncm-tcp-host/port for endpoint configuration option and use it for legacy as well (BACKEND-159) • add temperature sensor for CRTM Gateway and GRB 3000 (BACKEND-159) • add calibration command for CRTM 3000 LoRa devices (BACKEND-119) • add acceleration data sensor for CRTM 3000 Sensor (BACKEND-180) • add error code for temperature-sensor failures (‘DNE_FIRMWARE_TEMPERATURE‘) (BACKEND-180) • add fw-wcpu version for ZKL 3000, ZKL 3000 RC, ZKL 3000 RCC (BACKEND-136) • add ‘DNE_FIRMWARE_MEASUREMENT, DNE_FIRMWARE_GPS, DNE_FIRMWARE_MCU_COMM‘ (BACKEND-165) • device: Add new WUM sensor ‘wu-alarm-type‘ (TWS-125) • device: DNCM add BER sensor, and fw-modem version key (DINET-29) • Add support for NB-IoT CRM-3000 sensor (BACKEND-13) • tws-3000 dum & duu: Replace du-manual and du-ultrasonic with du-counter (TWS-178) • tws-3000 dum & duu: Add du-strike-role (BACKEND-253) • error codes: Add DNE_FIRMWARE_DU_STRIKE_ROLE_UNKNOWN (TWS-176) • error codes: Add DNE_FIRMWARE_MODEM_UART_FRAMING (DINET-64) • sensor data: Add support for multiple values in a single sensor:data (SVS-17)

Version	Date	Changes
		• add more specific firmware error battery levels (critical, empty, removed) (BACKEND-255) • add more specific firmware warning battery levels (critical, empty, removed) (BACKEND-255) • error code: Add DNE_FIRMWARE_NO_MEASUREMENT (BACKEND-175) • add error code DNE_WRN_BACKEND_DEVICE_WARNING (BACKEND-238) • add ‘"rt"‘ and ‘"rt:seqnr"‘ header property for Realtime message routing (TWS-190) • add ‘"time:recv"‘ header property for reply and publish messages to calculate the time delta (TWS-190)

Review list

REV = reviewer, VER = verifier, VAL = validator, AUT = author

Name	3.0.0	3.1.0	3.2.0	3.3.0
ing. J.J.J. Jacobs	AUT	AUT	AUT	AUT
ing. R.W.A. van der Heijden		AUT		AUT
ing. J.C.M. Raats		AUT	AUT	AUT
ir. R.H. van Lieshout				AUT
R.R.R. van Leeuwen				AUT

5 ## 1.2. Appendices

[1]	Title:	The application/json Media Type for JavaScript Object Notation (JSON)
	Author(s):	D. Crockford, JSON.org
	Version:	RFC 4627, July 2006
	File/URL:	http://www.ietf.org/rfc/rfc4627.txt [Accessed 3 November 2016]
[2]	Title:	JSON-RPC 2.0
	Author(s):	JSON-RPC Working Group
	Version:	2.0 (2013-01-04)
	File/URL:	http://www.jsonrpc.org/specification
[3]	Title:	Semantic Versioning
	Author(s):	Tom Preston-Werner
	Version:	2.0.0 (2013-06-18)
	File/URL:	http://semver.org/spec/v2.0.0.html [Accessed 3 November 2016]
[4]	Title:	ISO 8601
	Author(s):	ISO Technical Committee TC 154
	Version/Date:	ISO 8601:2004 (1 December 2004)
	File/URL:	http://en.wikipedia.org/wiki/ISO_8601 [Accessed 3 November 2016]
[5]	Title:	Redis NoSQL
	Author(s):	Redis Labs
	File/URL:	http://redis.io [Accessed 3 November 2016]
[6]	Title:	Key words for use in RFCs to Indicate Requirement Levels
	Authors:	S. Bradner
	Version/Date:	RFC 2119, March 1997
	File/URL:	http://www.ietf.org/rfc/rfc2119.txt [Accessed 3 November 2016]
[7]	Title:	Uniform Resource Identifier (URI): Generic Syntax
	Authors:	T. Berners-Lee
	Version/Date:	RFC 3986, January 2005
	File/URL:	http://www.ietf.org/rfc/rfc3986.txt [Accessed 14 June 2018]

1.3. Definitions and Abbreviations

1.3.1. Definitions

Text marking

Marked text	Text needs to be changed or completed.
Marked text	Text has changed compared to the previous release.
Marked section	Section headers that are intended for review.

5 Numbers

'*a*'	Numeric binary notation (*a* can be multiple 0s or 1s). E.g. '`010`' is a 3-bit value representing the binary number two. This kind of notation implies a specific bit length.
'*aa.aaaa*'	Numeric binary notation with '.' separations for clear reading of long binary numbers.
0x*a*	Numeric hexadecimal notation (*a* can be a digit 0 through 9, A through F). E.g. '`0x1A`' is hexadecimal number twenty-six. This kind of notation does not directly imply a bit length.
0x*aa.aaaa*	Numeric hexadecimal notation with '.' separations for clear reading of long hexadecimal numbers.
*a*d	Numeric (explicit) decimal notation. This kind of notation does not directly imply a bit length.
X[b:a]	Vector notation for vector X with bit range b downto a (little endian notation).

1.3.2. Abbreviations

AES-128	Advanced Encryption Standard (128-bit)
CBC	Cipher Block Chaining
CCPSKE	Challenged Cryptographic Pre-Shared Key Exchange
DI-Net	Dual Inventive Network
DNCM	DI-Net Communication Module
HMAC	Hash-based Message Authentication Code
IVFI	Initialization Value Fibonacci Interleaving
JSON	JavaScript Object Notation
MD5	Message digest 5
RPC	Remote Procedure Call
RT	Realtime
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
di-smp	DI-Net Secure Multi Proxy

Contents

2. Introduction

DI-Net-RPC is a stateless, light-weight remote procedure call (RPC) protocol. The protocol is heavily influenced by JSON-RPC 2.0. It describes the communication of request/reply and publish/subscribe packets between Devices and Clients. Primarily this specification defines several data structures and the rules around their processing. It is transport agnostic in that the concepts can be used within the same process, over sockets, over http, or in many various message passing environments. It uses JSON as data format.

On constrained systems where memory and connection throughput is limited, JSON is not considered light-weight. The MessagePack protocol solves this by compressing JSON types into a binary format. MessagePack is JSON compatible with the exception of the binary type extension and MessagePack allows other types than string as key for an Object.

2.1. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in `RFC 2119`.

Since DI-Net-RPC utilizes JSON, it has the same type system (see `http://www.json.org` or `RFC 4627`). JSON can represent four primitive types (Strings, Numbers, Booleans, and Null) and two structured types (Objects and Arrays). The term "Primitive" in this specification references any of those four primitive JSON types. The term "Structured" references either of the structured JSON types. Whenever this document refers to any JSON type, the first letter is always capitalized: Object, Array, String, Number, Boolean, Null. True and False are also capitalized.

All member names exchanged between the Client and the Device that are considered for matching of any kind should be considered to be case-sensitive. The terms function, method, and procedure can be assumed to be interchangeable.

The Client is defined as the origin of Request objects and the handler of Response objects. The Device is defined as the origin of Response objects and the handler of Request objects.

One implementation of this specification could easily fill both of those roles, even at the same time, to other different clients or the same client. This specification does not address that layer of complexity.

2.2. Device resources

Device resources define which Device resource is accessed or published. The value MUST be an String. It is applicable to the rpc object members `"req"`, `"rep"`, `"pub"`.

3. RPC objects

The protocol distinguishes multiple different design-patterns into seperated communication objects.

3.1. Request object

A rpc call is represented by sending a Request object. The Request object has the following members:

dinetrpc

A Number specifying the version of the DI-Net-RPC protocol. MUST be exactly 1.

device:uid

This member is REQUIRED when there is no `"project:id"` and `"user:id"` member. This member MUST be OMITTED if a `"project:id"` is specified. This member MAY coexist with the member `"user:id"`. See device:uid.

project:id

This member is REQUIRED when there is no "`device:uid`" and "`user:id`" member. This member MUST be OMITTED if a "`device:uid`" is specified. This member MAY coexist with the member "`user:id`". See project:id.

user:id

This member MAY coexist with the "`device:uid`" or "`project:id`" member. This member is RE↩QUIRED for the secure server frontend API, see secure server. And is OPTIONAL for all other messages.

req

A String specifying "`[classname]:[methodname]`". Both classname and methodname MUST only contain a-z. The classname MUST be less or equal to 10 characters. The methodname (including possible submethods) must be less or equal to 20 characters.

async

- This member is OPTIONAL.
- A String specifying "`queue`" or "`status`".
- When this member is omitted or contains something else then defined the request is considered synchronous, see asynchronous request-reply.

time

- This member is REQUIRED.
- Describes timestamp when the message was transmitted. See time data type.

params

A Structered value that holds the parameter value to be used during the invocation of the request. This member is OPTIONAL but MAY also have a Null value.

id

An identifier established by the Client that MUST contain a Number without fractional part and is REQUIRED for the request. The size of this identifier is unsigned 32-bit integer. Zero is considered valid for the identifier.

For every request the id MUST be different. Even when the error code `DNE_AGAIN` is returned on reply.

3.1.1. Parameter Structures

If present, parameters for the rpc call MUST be provided as a Structured value. Either by-position through an Array or by-name through an Object.

- by-position: params MUST be an Array, containing the values in the Device expected order.
- by-name: params MUST be an Object, with member names that match the Device expected parameter names. The absence of expected names MAY result in an error being generated. The names MUST match exactly, including case, to the method's expected parameters.

3.2. Reply object

When a RPC call is made, the Device MUST reply with a Response, except for in the case of Notifications. The Response is expressed as a single Object, with the following members:

dinetrpc

A Number specifying the version of the DI-Net-RPC protocol. MUST be exactly 1.

device:uid

This member is REQUIRED when there is no "`project:id`" member. See device:uid. This member MAY coexist with the member "`user:id`".

project:id

This member is REQUIRED when there is no `"device:uid"` member. See project:id. This member MAY coexist with the member `"user:id"`.

user:id

This member is REQUIRED for the secure server frontend API, see secure server. And is OPTIONAL for all other messages. This member MAY coexist with the `"device:uid"` or `"project:id"` member.

rep

A String specifying `"[classname]:[methodname]"`. Both classname and methodname MUST only contain a-z. The classname MUST be less or equal to 10 characters. The methodname (including possible submethods) must be less or equal to 20 characters.

async

- This member is OPTIONAL.
- A String specifying `"queue"` or `"status"`.
- When this member is omitted the reply is for a synchronous request, see asynchronous request-reply.

time

- This member is REQUIRED.
- Describes timestamp when the message was transmitted. See time data type.

time:recv

- This member is OPTIONAL.
- Describes timestamp when the reply message was received. See time data type.

id

- This member is REQUIRED.
- It MUST be the same as the value of the id member in the Request Object.
- If there was an error detecting the id in the Request object (e.g Parse error/Invalid Request), it MUST be Null.

result

- This member is OPTIONAL on success. This member MUST be an Array when present.
- This member MUST NOT exist if there was an error invoking the request.

error

- This member is REQUIRED on error.
- This member MUST NOT exist if there was no error triggered during invocation.
- The value for this member MUST be an Error object.

3.2.1. Error reply

When a RPC call encounters an error, the Response Object MUST contain the error member with a value that is a Object with the following members:

code

A Number that indicates the error type that occurred. See error codes.

descr

A String providing a short description of the error. The message SHOULD be in English and limited to a concise single sentence.

data

A Primitive or Structured value that contains additional information about the error. This MAY be omitted. The value of this member is defined by the Device (e.g. detailed error information, nested errors etc.).

3.3. Publish object

The Publish object is used to transmit information without a request. The Publish is expressed as a single Object, with the following members:

dinetrpc

A Number specifying the version of the DI-Net-RPC protocol. MUST be exactly 1.

device:uid

This member is REQUIRED when there is no `"project:id"` member. See device:uid.

project:id

This member is REQUIRED when there is no `"device:uid"` member. See project:id.

rt

- This member is OPTIONAL.
- Mutual exclusive with **rt:seqnr** property
- Property value is an Boolean set to `true` (`false` is invalid, as all messages are default non-↩ Realtime)

The Realtime flag is used to mark the message for Realtime but unreliable routing (e.g over UDP).

rt:seqnr

- This member is OPTIONAL.
- Mutual exclusive with **rt** property
- Property value is a Number within uint16 range which MAY overflow

The Realtime message sequence number for message (de)duplication when send published over multiple channels.

pub

A String specifying `"[classname]:[methodname]"`. Both classname and methodname MUST only contain a-z. The classname MUST be less or equal to 10 characters. The methodname (including possible submethods) must be less or equal to 20 characters.

time

- This member is REQUIRED.
- Describes timestamp when the message was published. See time data type.

time:recv

- This member is OPTIONAL.
- Describes timestamp when the publish message was received. See time data type.

result

This member is OPTIONAL. And MUST be an Array or a Structure. The structure of the object is specified by the corresponding class method properties sections.

4. Data types

Type	Description	Zero value
"bool"	A Boolean or Number.	0 or false
"number"	A Number which MAY contain a fractional part.	-1 or -1.0, 0 or 0.0
"numbers"	An array of "number" elements	[] or null
"enum"	Enumerated type as Number.	See enum
"gps"	GPS location and information. This MUST be a JSON object.	See gps
"struct"	composed datatype. Fields MUST be of a primitive type.	See struct
"sw_version"	DI-Net software build version. This MUST be a String.	See version
"time"	DI-Net timestamp. This member MUST be a Number without fractional part	See time
"errno"	DI-Net error number (enum di_errno).	See errno
"error"	DI-Net error sample	See error

4.1. Precision

When the number type is used a precision SHOULD be specified.

- u8, unsigned integer 8-bit
- u16, unsigned integer 16-bit
- u32, unsigned integer 32-bit
- u64, unsigned integer 64-bit
- s8, signed integer 8-bit
- s16, signed integer 16-bit
- s32, signed integer 32-bit
- s64, signed integer 64-bit
- float, single precision floating point
- double, double precision floating point

This directly maps to the C/C++ stdint implementation.

4.2. Enumeration

Enumerated type is a Number remapped to a string definition. When enumerated type information is requested the key "enum" is added and value MUST be an Object and contains key-values where value MUST be an number. Key strings MUST be lower-case. E.g:

```
{
    "type" : "enum",
    "enum" :
    {
        "foo" : 0,
        "bar" : 1
    }
}
```

4.3. Struct

Struct type is composed of one or more key-values. Each field must be of type datatype, but cannot be of type stuct. In other words, structs cannot be nested. E.g ZKL 3000 RC short/battery state struct:

```
{
    "value" : {
        "state" : true,
        "section_1" : true,
        "section_2" : true,
        "section_3" : true,
        "section_4" : true
    }
}
```

4.4. Time

The `time` datatype, represents the time in UTC with millisecond granularity since UNIX epoch. It will not count leap seconds.

- This value is UNIX timestamp * 1000 (epoch time starts at 1 januari 1970 00:00:00.000)
- SHOULD be represented as 64-bit unsigned integer

4.5. Versioning 'version'

All DI-Net software version descriptions is formatted as follows:

- Format: `"[semantic version]+[git commit short sha1 hash]+[build datetime iso8601 utc]"`.
- Formatted as `"AA.BB.CC+YYYYmmddhhMMss+git.GGGGGGG"` where:
  - AA: Semantic major version (leading zero MAY be omitted)
  - BB: Semantic minor version (leading zero MAY be omitted)
  - CC: Semantic patch version (leading zero MAY be omitted)
  - YYYY: Year of build
  - mm: Month of build
  - dd: Day of build
  - hh: Hour of build
  - MM: Minute of build
  - ss: Second of build
  - GGGGGGG: Git commit short sha1 hash
- Example `"0.2.0+20160929122208+git.64ac14a-dirty"`:
  - Semantic version : `"0.2.0"` (Major 0, Minor 2, Patch 0)
  - Build date at time: 2016-09-29 at 12:22:08 UTC
  - Git commit: `"64ac14a"`
  - Uncommitted git changes: "-dirty"

For release builds `"[semantic version]"` points at a version tag and no extra metadata is supplied.

One or multiple versions must be formated as a Struct. Keys must be formatted as follows:

- `<prefix>-<component> : <version>`
  - `<prefix>` MUST be one of:
    - `hw`: Hardware (PCB) version, formatted as `<major>.<minor>`
    - `fw`: Firmware version, formatted in compliance with semver.org without `v` prefix. For 3thparty components it MAY be a freeform string (e.g `"RHL769x.2.23.172400.↩ 201706231140.x7120m_1"`).
  - `<component>` string should only contain characters in the `[a-z0-9_]` regex

Example

```
{
    "hw-main": "1.2",
    "hw-switch": "1.3",

    "fw-main": "1.0.2",
    "fw-switch_control" : "1.0.3",
    "fw-switch_meas" : "1.2.1",
    "fw-switch_drive" : "1.0.2",
}
```

4.6. GPS

The GPS information MUST be a object with the following members:

Member	Type	Example	Description
`"latitude"`	Number (double)	51.585736	Latitude as decimal value
`"longitude"`	Number (double)	5.195319	Longitude as decimal value
`"hdop"`	Number (float)	3.8	Horizontal dilution of precision (DOP)

4.7. Error object

A single error sample when one or more errors of the same number are detected. They MAY be send periodic.

Member	Type	Example	Description
`"code"`	`errno`	`DNE_FIRMWARE_TILT`	DI-Net error number
`"count"`	number (u32)	10	Error counter
`"first"`	time	1475672389000	DI-Net time when first error is reported (Wed Oct 5 12:59:49 UTC 2016)
`"last"`	time	1475672461000	DI-Net time when last error is reported (Wed Oct 5 13:01:01 UTC 2016)

5. Error codes

When a RPC call encounters an error, the resulting error MUST contain an error from the list below. The number that indicates the error type that occurred MUST be an positive 32-bit unsigned integer (`UINT32_MAX - 1`).

The DI-Net error codes are divided in ranges. The code 0 (zero) is reserved for OK like POSIX and defined as enumerator `DNOK`. And `UINT32_MAX` is reserved for unknown/reserved errors.

Range	Category
1 - 255	Generic
256 - 511	Secure server
512 - 767	Business-logic
768 - 1023	Reserved for future use
1024 - 2999	Firmware and devices
3000 - 3050	CAN protocol
3051 - 9215	Reserved for future use
9216 - 10239	Frontend

10240 - 999999	Reserved for future use
1000000 - 1000999	MTinfo 3000 client API
1001000 - 1999999	Reserved for future use
2000000 - 2000999	[Firmware and device warnings]()
2001000 - 2001255	[Business-logic warnings]()
2001256 - (UINT32_MAX - 1)	Reserved for future use
`UINT32_MAX`	Unknown/reserved

5.1. Generic

Code	Enumerator	Description
1	`DNE_PARAM`	Invalid parameter
2	`DNE_RANGE`	Value/parameter out of range
3	`DNE_NODEV`	No such device
4	`DNE_NORES`	No such resource
5	`DNE_BUSY`	Device or resource busy
6	`DNE_OPNOTSUPP`	Operation not supported
7	`DNE_PROTO`	Protocol error
8	`DNE_CHECKSUM`	Checksum error
9	`DNE_TIMEOUT`	Timeout
10	`DNE_DISCONNECTED`	Disconnected
11	`DNE_AGAIN`	Try again
12	`DNE_OPDENIED`	Operation denied
13	`DNE_IOFAILED`	I/O operation failed
14	`DNE_NOTFOUND`	Not found
15	`DNE_NOMEM`	Out of memory
16-255	-	Reserved for future use

5.2. Secure server

Code	Enumerator	Description
256	`DNE_DUPLICATE_PROJECT`	Duplicate project found
257	`DNE_INVALID_PROJECT_STATE`	Invalid project state reached
258	`DNE_INVALID_MUTATION`	Invalid mutation requested
259	`DNE_INVALID_ACTION`	Invalid action specified
260	`DNE_PROJECT_CHANGED`	Project data changed
261	`DNE_PLANNING_CHANGED`	Project planning changed
262	`DNE_DESIGN_CHANGED`	Project design changed
263	`DNE_DATABASE_ERROR`	Database error occurred
264	`DNE_NO_PROJECT_DATA`	No project data supplied
265	`DNE_INVALID_PROJECT`	Invalid project selected
266	`DNE_INVALID_REQUEST`	Invalid request supplied
267	`DNE_INTERNAL_ERROR`	Internal error supplied
268	`DNE_NO_BACKEND`	Backend is not connected
269	`DNE_BACKEND_INVALID_MESSA↩ GE`	Backend sent an invalid message
270	`DNE_SETUP_CHANGED`	Project setup changed
271	`DNE_INVALID_PROJECT_TYPE`	Project type not recognized
272	-	Reserved for future use
273	`DNE_NO_PROJECT_GROUP`	No project groups provided
274-511	-	Reserved for future use

5.3. Business-logic

Code	Enumerator	Description
512	DNE_BACKEND_DUPLICATE_PRO↩ JECT	Duplicate project
513	DNE_BACKEND_INVALID_PROJE↩ CT	Invalid project
514	DNE_BACKEND_DEVICE_ERROR	Device error
515	DNE_BACKEND_INTERNAL_ERROR	Backend internal error
516	DNE_BACKEND_INVALID_MUTAT↩ ION	Invalid mutation requested
517	DNE_BACKEND_INVALID_GROUP	Invalid group action
518-767	-	Reserved for future use

5.4. Firmware/device

Code	Enumerator	Description
1024	DNE_FIRMWARE_TILT	Tilt position error
1025	DNE_FIRMWARE_ULTRASONIC	Ultrasonic error
1026	DNE_FIRMWARE_SPEAKER	Speaker error
1027	DNE_FIRMWARE_EEPROM	EEPROM error
1028	DNE_FIRMWARE_CALIBRATION	Calibration error
1029	DNE_FIRMWARE_BA	BA error
1030	DNE_FIRMWARE_VREF	Vref error
1031	DNE_FIRMWARE_VREF_CLIPPING	Vref clipping
1032	DNE_FIRMWARE_LEADER_FAILU↩ RE	Leader failed (failsafe)
1033	DNE_FIRMWARE_FOLLOWER_FAI↩ LURE	Follower failed (failsafe)
1034	DNE_FIRMWARE_ROTATION	Rotation position error
1035	DNE_FIRMWARE_DETECTIONBUT↩ TON	Detection button error
1036	DNE_FIRMWARE_DEADMANS_TIM↩ EOUT	Deadmans timeout
1037	DNE_FIRMWARE_BATTERY_TOO_↩ LOW	One or more device batteries are too low
1038	DNE_FIRMWARE_SWITCH_COMM	Switch communication error
1039	DNE_FIRMWARE_LEADER_CONFL↩ ICT	Leader conflict
1040	DNE_FIRMWARE_TEMPERATURE	Temperature sensor failure
1041	DNE_FIRMWARE_MEASUREMENT	Measurement error
1042	DNE_FIRMWARE_GPS	GPS error
1043	DNE_FIRMWARE_MCU_COMM	MCU communication error
1044	DNE_FIRMWARE_DU_STRIKE_RO↩ LE_UNKNOWN	Detection unit strike role unknown
1045	DNE_FIRMWARE_MODEM_UART_F↩ RAMING	Modem UART framing error
1046	DNE_FIRMWARE_BATTERY_BOARD	At least 1 battery section broken
1047	DNE_FIRMWARE_SPEAKER_L	Speaker error Main board
1048	DNE_FIRMWARE_SPEAKER_F	Speaker error Monitor board
1049	DNE_FIRMWARE_EEPROM_L	EEPROM error Main board

1050	DNE_FIRMWARE_EEPROM_F	EEPROM error Monitor board
1051	DNE_FIRMWARE_BATTERY_TOO_↩ LOW_L	Battery too low Main board
1052	DNE_FIRMWARE_BATTERY_TOO_↩ LOW_F	Battery too low Monitor board
1053	DNE_FIRMWARE_BATTERY_CRIT↩ ICAL	Battery is critical
1054	DNE_FIRMWARE_BATTERY_EMPTY	Battery is empty
1055	DNE_FIRMWARE_BATTERY_REMO↩ VED	Battery is removed
1056	DNE_FIRMWARE_NO_MEASUREME↩ NT	There is no measurement available
1057 - 2999	-	Reserved for future use

5.5. CAN-bus and protocol

Code	Enumerator	Description
3000	DNE_CAN_INVAL	CAN invalid argument
3001	DNE_CAN_IO	CAN peripherial I/O error
3002	DNE_CAN_TIMEOUT	CAN timeout
3003	DNE_CAN_AGAIN	CAN try again
3004	DNE_CAN_NOBUFS	CAN buffers depleted
3005	DNE_CAN_PROTO	CAN protocol error
3006	DNE_CAN_CRC	CAN msg crc error

5.6. Frontend

Code	Description
9216 - 9299	Reserved for future use
9300	Invalid MTI websocket server token (mti-wss)
9301 - 10239	Reserved for future use

5.7. Firmware/device warnings

Code	Enumerator	Description
2000000	DNE_WRN_FIMRWARE_SHORT_PR↩ ESENT	Short already present
2000001	DNE_WRN_FIRMWARE_BATTERY_↩ CRITICAL	Battery is critical
2000002	DNE_WRN_FIRMWARE_BATTERY_↩ EMPTY	Battery is empty
2000003	DNE_WRN_FIRMWARE_BATTERY_↩ REMOVED	Battery is removed
2000004 - 2000999	-	Reserved for future use

5.8. Business-logic warnings

Code	Enumerator	Description
2001000	DNE_WRN_BACKEND_DEVICE_WA↩ RNING	Device warning
2001001 - 2001255	-	Reserved for future use

6. Messsaging proxy

The Messaging proxy only communicates messages. Therefore the message class applies here. Some fields are only available for a few platforms. See per field a detailed overview.

6.1. Title & Message

The title and message are used in the notification itself. When provided a notification is visible on the mobile. When both are set the title is formatted in bold.

6.2. Icon

Only available when the title or message is set.

- UWP: The icon is an uri starting with `http(s)://`, `ms-appx:///` or `ms-appdata:///local/` that points to an image. On windows phone 8.1 this image is used to replace the app's logo.
- Android: The icons can be found in the drawable resource.
- iOS: not supported.

6.3. Sound

Only available when the title or message is set.

- UWP: not supported.
- Android: the sound to play when the device receives the notification. The sound must reside in `/res/raw/`.
- iOS: the sound to play when the device receives the notifiation. The sound files can be in the main bundle of the client app or in the Library/Sounds folder of the app's data container.

6.4. Tag

Only available when the title or message is set.

- UWP: not supported.
- Android: identifier used to replace existing notifications in the notification drawer.
- iOS: not supported.

6.5. Action

Only available when the title or message is set.

- UWP: not supported.
- Android: the action associated with a user click on the notification. If specified, an activity with a matching intent filter is launched when a user clicks on the notification.
- iOS: not supported.

6.6. Data

Custom data (in JSON format) to send to the phone.

- UWP: only available when the title, message and badge is not set.
- Android: always available
- iOS: always available

6.7. Collapsekey

Only available when the title or message is set.

- UWP: not supported.
- Android: This parameter identifies a group of messages that can be collapsed, so that only the last message gets sent when delivery can be resumed.
- iOS: not supported.

6.8. Priority

The priority to send this push message.

- UWP: not supported.
- Android: `"normal"` for normal push messages or `"high"` for imported push messages
- iOS: not supported.

6.9. Badge

The value of the badge on the home screen app icon. If not specified, the badge is not changed. If set to `0`, the badge is removed.

- UWP: only available when the title and message is not set.
- Android: always available
- iOS: always available

7. MySQL proxy

The MySQL uses the following RPC class:methods

- user:data
- device:user:data
- translate:get

8. MessagePack specification

NOTE: This is a copy of `https://github.com/msgpack/msgpack/blob/master/spec.md`

commit 0b8f5ac67cdd130f4d4d4fe6afb839b989fdb86a at Dec 22, 2015 (with patched changes to representation).

MessagePack is an object serialization specification like JSON.

MessagePack has two concepts: **type system** and **formats**.

Serialization is conversion from application objects into MessagePack formats via MessagePack type system.

Deserialization is conversion from MessagePack formats into application objects via MessagePack type system.

```
Serialization:
    Application objects
    -->  MessagePack type system
    -->  MessagePack formats (byte array)

Deserialization:
    MessagePack formats (byte array)
    -->  MessagePack type system
    -->  Application objects
```

This document describes the MessagePack type system, MessagePack formats and conversion of them.

8.1. Type system

- Types
    - **Integer** represents an integer
    - **Nil** represents nil
    - **Boolean** represents true or false

- **Float** represents a IEEE 754 double precision floating point number including NaN and Infinity
- **Raw**
  - **String** extending Raw type represents a UTF-8 string
  - **Binary** extending Raw type represents a byte array
- **Array** represents a sequence of objects
- **Map** represents key-value pairs of objects
- **Extension** represents a tuple of type information and a byte array where type information is an integer whose meaning is defined by applications

8.1.1. Limitation

- a value of an Integer object is limited from `-(2^63)` upto `(2^64)-1`
- maximum length of a Binary object is `(2^32)-1`
- maximum byte size of a String object is `(2^32)-1`
- String objects may contain invalid byte sequence and the behavior of a deserializer depends on the actual implementation when it received invalid byte sequence
  - Deserializers should provide functionality to get the original byte array so that applications can decide how to handle the object
- maximum number of elements of an Array object is `(2^32)-1`
- maximum number of key-value associations of a Map object is `(2^32)-1`

Extension type

MessagePack allows applications to define application-specific types using the Extension type. Extension type consists of an integer and a byte array where the integer represents a kind of types and the byte array represents data.

Applications can assign `0` to `127` to store application-specific type information.

MessagePack reserves `-1` to `-128` for future extension to add predefined types which will be described in separated documents.

```
[0, 127]: application-specific types
[-128, -1]: reserved for predefined types
```

8.2. Formats

Overview

format name	first byte (in binary)	first byte (in hex)
positive fixint	0xxxxxxx	0x00 - 0x7f
fixmap	1000xxxx	0x80 - 0x8f
fixarray	1001xxxx	0x90 - 0x9f
fixstr	101xxxxx	0xa0 - 0xbf
nil	11000000	0xc0
(never used)	11000001	0xc1
false	11000010	0xc2
true	11000011	0xc3

bin 8	11000100	0xc4
bin 16	11000101	0xc5
bin 32	11000110	0xc6
ext 8	11000111	0xc7
ext 16	11001000	0xc8
ext 32	11001001	0xc9
float 32	11001010	0xca
float 64	11001011	0xcb
uint 8	11001100	0xcc
uint 16	11001101	0xcd
uint 32	11001110	0xce
uint 64	11001111	0xcf
int 8	11010000	0xd0
int 16	11010001	0xd1
int 32	11010010	0xd2
int 64	11010011	0xd3
fixext 1	11010100	0xd4
fixext 2	11010101	0xd5
fixext 4	11010110	0xd6
fixext 8	11010111	0xd7
fixext 16	11011000	0xd8
str 8	11011001	0xd9
str 16	11011010	0xda
str 32	11011011	0xdb
array 16	11011100	0xdc
array 32	11011101	0xdd
map 16	11011110	0xde
map 32	11011111	0xdf
negative fixint	111xxxxx	0xe0 - 0xff

8.2.1. Notation in diagrams

```
one byte:
+--------+
|        |
+--------+


a variable number of bytes:
+========+
|        |
+========+


variable number of objects stored in MessagePack format:
+~~~~~~~~~~~~~~~~~+
|                 |
+~~~~~~~~~~~~~~~~~+
```

`X`, `Y`, `Z` and `A` are the symbols that will be replaced by an actual bit.

8.2.2. nil format

Nil format stores nil in 1 byte.

```
nil:
+--------+
|  0xc0  |
+--------+
```

8.2.3. bool format family

Bool format family stores false or true in 1 byte.

```
false:
+--------+
|  0xc2  |
+--------+


true:
+--------+
|  0xc3  |
+--------+
```

8.2.4. int format family

Int format family stores an integer in 1, 2, 3, 5, or 9 bytes.

```
positive fixnum stores 7-bit positive integer
+--------+
|0XXXXXXX|
+--------+


negative fixnum stores 5-bit negative integer
+--------+
|111YYYYY|
+--------+

* 0XXXXXXX is 8-bit unsigned integer
* 111YYYYY is 8-bit signed integer

uint 8 stores a 8-bit unsigned integer
+--------+--------+
|  0xcc  |ZZZZZZZZ|
+--------+--------+


uint 16 stores a 16-bit big-endian unsigned integer
+--------+--------+--------+
|  0xcd  |ZZZZZZZZ|ZZZZZZZZ|
+--------+--------+--------+


uint 32 stores a 32-bit big-endian unsigned integer
+--------+--------+--------+--------+--------+
|  0xce  |ZZZZZZZZ|ZZZZZZZZ|ZZZZZZZZ|ZZZZZZZZ|
+--------+--------+--------+--------+--------+


uint 64 stores a 64-bit big-endian unsigned integer
+--------+--------+--------+--------+--------+--------+--------+--------+--------+
|  0xcf  |ZZZZZZZZ|ZZZZZZZZ|ZZZZZZZZ|ZZZZZZZZ|ZZZZZZZZ|ZZZZZZZZ|ZZZZZZZZ|ZZZZZZZZ|
+--------+--------+--------+--------+--------+--------+--------+--------+--------+


int 8 stores a 8-bit signed integer
+--------+--------+
|  0xd0  |ZZZZZZZZ|
+--------+--------+


int 16 stores a 16-bit big-endian signed integer
+--------+--------+--------+
|  0xd1  |ZZZZZZZZ|ZZZZZZZZ|
+--------+--------+--------+


int 32 stores a 32-bit big-endian signed integer
+--------+--------+--------+--------+--------+
|  0xd2  |ZZZZZZZZ|ZZZZZZZZ|ZZZZZZZZ|ZZZZZZZZ|
+--------+--------+--------+--------+--------+
int 64 stores a 64-bit big-endian signed integer
+--------+--------+--------+--------+--------+--------+--------+--------+--------+
|  0xd3  |ZZZZZZZZ|ZZZZZZZZ|ZZZZZZZZ|ZZZZZZZZ|ZZZZZZZZ|ZZZZZZZZ|ZZZZZZZZ|ZZZZZZZZ|
+--------+--------+--------+--------+--------+--------+--------+--------+--------+
```

8.2.5. float format family

Float format family stores a floating point number in 5 bytes or 9 bytes.

```
float 32 stores a floating point number in IEEE 754 single precision floating point number format:
+--------+--------+--------+--------+--------+
|  0xca  |XXXXXXXX|XXXXXXXX|XXXXXXXX|XXXXXXXX|
+--------+--------+--------+--------+--------+


float 64 stores a floating point number in IEEE 754 double precision floating point number format:
+--------+--------+--------+--------+--------+--------+--------+--------+--------+
|  0xcb  |YYYYYYYY|YYYYYYYY|YYYYYYYY|YYYYYYYY|YYYYYYYY|YYYYYYYY|YYYYYYYY|YYYYYYYY|
+--------+--------+--------+--------+--------+--------+--------+--------+--------+
```

Where:

- XXXXXXXX_XXXXXXXX_XXXXXXXX_XXXXXXXX is a big-endian IEEE 754 single precision floating point number. Extension of precision from single-precision to double-precision does not lose precision.
- YYYYYYYY_YYYYYYYY_YYYYYYYY_YYYYYYYY_YYYYYYYY_YYYYYYYY_YYYYYYYY_↩ YYYYYYYY is a big-endian IEEE 754 double precision floating point number

8.2.6. str format family

Str format family stores a byte array in 1, 2, 3, or 5 bytes of extra bytes in addition to the size of the byte array.

```
fixstr stores a byte array whose length is upto 31 bytes:
+--------+========+
|101XXXXX|  data  |
+--------+========+


str 8 stores a byte array whose length is upto (2^8)-1 bytes:
+--------+--------+========+
|  0xd9  |YYYYYYYY|  data  |
+--------+--------+========+


str 16 stores a byte array whose length is upto (2^16)-1 bytes:
+--------+--------+--------+========+
|  0xda  |ZZZZZZZZ|ZZZZZZZZ|  data  |
+--------+--------+--------+========+


str 32 stores a byte array whose length is upto (2^32)-1 bytes:
+--------+--------+--------+--------+--------+========+
|  0xdb  |AAAAAAAA|AAAAAAAA|AAAAAAAA|AAAAAAAA|  data  |
+--------+--------+--------+--------+--------+========+
```

Where:

- XXXXX is a 5-bit unsigned integer which represents N
- YYYYYYYY is a 8-bit unsigned integer which represents N
- ZZZZZZZZ_ZZZZZZZZ is a 16-bit big-endian unsigned integer which represents N
- AAAAAAAA_AAAAAAAA_AAAAAAAA_AAAAAAAA is a 32-bit big-endian unsigned integer which represents N
- N is the length of data

8.2.7. bin format family

Bin format family stores an byte array in 2, 3, or 5 bytes of extra bytes in addition to the size of the byte array.

```
bin 8 stores a byte array whose length is upto (2^8)-1 bytes:
+--------+--------+========+
|  0xc4  |XXXXXXXX|  data  |
+--------+--------+========+


bin 16 stores a byte array whose length is upto (2^16)-1 bytes:
+--------+--------+--------+========+
|  0xc5  |YYYYYYYY|YYYYYYYY|  data  |
+--------+--------+--------+========+


bin 32 stores a byte array whose length is upto (2^32)-1 bytes:
+--------+--------+--------+--------+--------+========+
|  0xc6  |ZZZZZZZZ|ZZZZZZZZ|ZZZZZZZZ|ZZZZZZZZ|  data  |
+--------+--------+--------+--------+--------+========+
```

Where:

- XXXXXXXX is a 8-bit unsigned integer which represents N
- YYYYYYYY_YYYYYYYY is a 16-bit big-endian unsigned integer which represents N
- ZZZZZZZZ_ZZZZZZZZ_ZZZZZZZZ_ZZZZZZZZ is a 32-bit big-endian unsigned integer which represents N
- N is the length of data

8.2.8. array format family

Array format family stores a sequence of elements in 1, 3, or 5 bytes of extra bytes in addition to the elements.

```
fixarray stores an array whose length is upto 15 elements:
+--------+~~~~~~~~~~~~~~~~~+
|1001XXXX|    N objects    |
+--------+~~~~~~~~~~~~~~~~~+


array 16 stores an array whose length is upto (2^16)-1 elements:
+--------+--------+--------+~~~~~~~~~~~~~~~~~+
|  0xdc  |YYYYYYYY|YYYYYYYY|    N objects    |
+--------+--------+--------+~~~~~~~~~~~~~~~~~+


array 32 stores an array whose length is upto (2^32)-1 elements:
+--------+--------+--------+--------+--------+~~~~~~~~~~~~~~~~~+
|  0xdd  |ZZZZZZZZ|ZZZZZZZZ|ZZZZZZZZ|ZZZZZZZZ|    N objects    |
+--------+--------+--------+--------+--------+~~~~~~~~~~~~~~~~~+
```

Where:

- XXXX is a 4-bit unsigned integer which represents N
- YYYYYYYY_YYYYYYYY is a 16-bit big-endian unsigned integer which represents N
- ZZZZZZZZ_ZZZZZZZZ_ZZZZZZZZ_ZZZZZZZZ is a 32-bit big-endian unsigned integer which represents N N is the size of a array

8.2.9. map format family

Map format family stores a sequence of key-value pairs in 1, 3, or 5 bytes of extra bytes in addition to the key-value pairs.

```
fixmap stores a map whose length is upto 15 elements
+--------+~~~~~~~~~~~~~~~~~+
|1000XXXX|   N*2 objects   |
```

```
+--------+~~~~~~~~~~~~~~~~~+


map 16 stores a map whose length is upto (2^16)-1 elements
+--------+--------+--------+~~~~~~~~~~~~~~~~~+
|  0xde  |YYYYYYYY|YYYYYYYY|   N*2 objects   |
+--------+--------+--------+~~~~~~~~~~~~~~~~~+


map 32 stores a map whose length is upto (2^32)-1 elements
+--------+--------+--------+--------+--------+~~~~~~~~~~~~~~~~~+
|  0xdf  |ZZZZZZZZ|ZZZZZZZZ|ZZZZZZZZ|ZZZZZZZZ|   N*2 objects   |
+--------+--------+--------+--------+--------+~~~~~~~~~~~~~~~~~+
```

Where:

- XXXX is a 4-bit unsigned integer which represents N
- YYYYYYYY_YYYYYYYY is a 16-bit big-endian unsigned integer which represents N
- ZZZZZZZZ_ZZZZZZZZ_ZZZZZZZZ_ZZZZZZZZ is a 32-bit big-endian unsigned integer which represents N
- N is the size of a map
- odd elements in objects are keys of a map
- the next element of a key is its associated value

8.2.10. ext format family

Ext format family stores a tuple of an integer and a byte array.

```
fixext 1 stores an integer and a byte array whose length is 1 byte
+--------+--------+--------+
|  0xd4  |  type  |  data  |
+--------+--------+--------+


fixext 2 stores an integer and a byte array whose length is 2 bytes
+--------+--------+--------+--------+
|  0xd5  |  type  |       data      |
+--------+--------+--------+--------+


fixext 4 stores an integer and a byte array whose length is 4 bytes
+--------+--------+--------+--------+--------+--------+
|  0xd6  |  type  |                data               |
+--------+--------+--------+--------+--------+--------+


fixext 8 stores an integer and a byte array whose length is 8 bytes
+--------+--------+--------+--------+--------+--------+--------+--------+--------+--------+
|  0xd7  |  type  |                                  data                                 |
+--------+--------+--------+--------+--------+--------+--------+--------+--------+--------+


fixext 16 stores an integer and a byte array whose length is 16 bytes
+--------+--------+--------+--------+--------+--------+--------+--------+--------+--------+
|  0xd8  |  type  |                                  data
+--------+--------+--------+--------+--------+--------+--------+--------+--------+--------+
+--------+--------+--------+--------+--------+--------+--------+--------+
                              data (cont.)                              |
+--------+--------+--------+--------+--------+--------+--------+--------+

ext 8 stores an integer and a byte array whose length is upto (2^8)-1 bytes:
+--------+--------+--------+========+
|  0xc7  |XXXXXXXX|  type  |  data  |
+--------+--------+--------+========+


ext 16 stores an integer and a byte array whose length is upto (2^16)-1 bytes:
```

```
+--------+--------+--------+--------+========+
|  0xc8  |YYYYYYYY|YYYYYYYY|  type  |  data  |
+--------+--------+--------+--------+========+
```

```
ext 32 stores an integer and a byte array whose length is upto (2^32)-1 bytes:
+--------+--------+--------+--------+--------+--------+========+
|  0xc9  |ZZZZZZZZ|ZZZZZZZZ|ZZZZZZZZ|ZZZZZZZZ|  type  |  data  |
+--------+--------+--------+--------+--------+--------+========+
```

Where:

- XXXXXXXX is a 8-bit unsigned integer which represents N
- YYYYYYYY_YYYYYYYY is a 16-bit big-endian unsigned integer which represents N
- ZZZZZZZZ_ZZZZZZZZ_ZZZZZZZZ_ZZZZZZZZ is a big-endian 32-bit unsigned integer which represents N
- N is a length of data
- type is a signed 8-bit signed integer
- type < 0 is reserved for future extension including 2-byte type information

8.3. Serialization: type to format conversion

MessagePack serializers convert MessagePack types into formats as following:

source types	output format
Integer	int format family (positive fixint, negative fixint, int 8/16/32/64 or uint 8/16/32/64)
Nil	nil
Boolean	bool format family (false or true)
Float	float format family (float 32/64)
String	str format family (fixstr or str 8/16/32)
Binary	bin format family (bin 8/16/32)
Array	array format family (fixarray or array 16/32)
Map	map format family (fixmap or map 16/32)
Extension	ext format family (fixext or ext 8/16/32)

If an object can be represented in multiple possible output formats, serializers SHOULD use the format which represents the data in the smallest number of bytes.

8.4. Deserialization: format to type conversion

MessagePack deserializers convert MessagePack formats into types as following:

source formats	output type
positive fixint, negative fixint, int 8/16/32/64 and uint 8/16/32/64	Integer
nil	Nil
false and true	Boolean
float 32/64	Float
fixstr and str 8/16/32	String
bin 8/16/32	Binary
fixarray and array 16/32	Array
fixmap map 16/32	Map
fixext and ext 8/16/32	Extension

9. Asynchronous Request-Reply

Asynchronous Request-Reply may not be implemented in all applications, to check whether this is supported check the reply for the `"async"` member.

Request flow

When issueing a request, send the request like normal but with the `"async"` set to `"queue"`. The message shall be checked for validity and verified if it can be handled by the application. If it accepts the asynchronous request a reply is sent without errors indicating it is scheduled for delivery.

For the reply, another request must be issued. Send a request with the same class and method to the request-reply socket. But with `"async"` set to `"status"`. When a reply is unavailable an error is returned. When the reply becomes available, a status request is replied without an error and containing the (optionally) requested data.

10. Business-logic

All business-logics receive messages from the Secure server. Likewise the secure server only the project class applies here.

Current available business-logics

- TWS 3000 (dinet/tws3000.git)
- ZKL 3000 RC (dinet/zkl3000rc.git)
- CRTM 3000 (dinet/crtm3000.git)

Error reply

When a business logic replies with an error, metadata for devices is also added. In the error object the application specific `data` property is used to notify the requester which devices (`device:uid`) failed with an error reply. It is structured as `{"device:uid" : errno, "device:uid" : errno ...}`.

Example:

```
{
    "dinetrpc":1,
    "id":87,
    "rep":"project:unlock",
    "time":1448373137142,
    "project:id":15,
    "error" : {
        "code" : 514,
        "descr" : "Device error",
         "data" : {
             "00133410270034001251343236363736" : 9,
             "0013341035004d000951343132353330" : 12
         }
    }
}
```

11. Business logic [CRTM3000]

The business logic for the CRTM3000 is used for sending SMS messages when a threshold is exceeded. The thresholds are saved in the option member of the CRTM devices

11.1. Option

Name	Description
`"thresholds"`	List of thresholds

11.2. Thresholds

Name	Description
"sensor:uid"	UID of the sensor
"value"	The threshold value
"type"	The type of threshold
"from"	The time in seconds from midnight (at timezone) when the threshold starts sending SMS messages. This value is REQUIRED.
"to"	The time in seconds from midnight (at timezone) when the threshold stops sending SMS messages. This value is REQUIRED.
"timezone"	The timezone that is used to determinate midnight. The timezone is a value from the IANA time zone database. See list of tz database time zones for a list of valid time zones. This value is REQUIRED.

Time

The time is measured in seconds from midnight (at timezone). It is allowed that the "to" value is less than the "from" value. For example: "from" 61200 (17:00 at timezone) and "to" 30600 (08:30 at timezone) will send SMS messages from 17:00 to 08:30 the next day. When the "from" and "to" value are the same, SMS messages are always sent.

11.3. Threshold type

The following threshold types are valid:

Type	Value
"Less then"	"lt"
"Greater then"	"gt"

Example

```
{
  "dinetrpc": 1,
  "time": 1470733748175,
  "id": 12,
  "project:id": 9,
  "user:id": 6,
  "req": "project:return",
  "params": {
    "project": {
      "setup": {
        "project:id": 9,
        "type": "crtm"
      },
      "planning": {
        "groups": [{
          "id" : 1,
          "operators":[
            {"user:id":1,"phone:alarm" : "+31687654321"},
            {"user:id":2,"phone:alarm" : "+31612345678"}
          ]
        }],
        "devices": []
      },
      "design": {
        "devices": [
          {
            "device:uid": "01000000000000000000000000000623",
            "name": "C001",
            "option": {
              "thresholds": [
                {
                  "sensor:uid": 100,
                  "value": 30.5,
                  "type": "lt",
                  "from": 30600,
                  "to": 61200,
                  "timezone": "Europe/Amsterdam"
                },
                {
                  "sensor:uid": 101,
                  "value": 32,
                  "type": "gt",
                  "from": 0,
                  "to": 0,
                  "timezone": "Europe/Amsterdam"
                }
              ]
            }
          }
        ]
      }
    }
    },
    "async": "queue"
  }
```

12. Classes

All DI-Net RPC messages are categorized in `classes` which contain `methods`. Method requests can have an optional `param` struct property. A request on a `class:method` MAY return a filled `result` property.

Every DI-Net message class supplies a set of methods, and properties. When a class has a `info` method it always returns static properties which SHALL NOT change during runtime (e.g device firmware version).

When a class has a `data` method it always returns non-static properties which MAY change during runtime (e.g `sensor:data`).

For the action, config, notify and sensor class there are items which MAY be controlled or monitored (capability items). The `info` method of the class returns all or a subset of the capabilities that can be controlled or monitored. Each item contains a least an `uid`, `label` and `type` property (additional fields may exist per class).

The `label` is a human readable label for the capability item. The `label` and `type` properties are paired together which MUST never be changed in releases of the RPC to remain backwards compatibility. The same `label` can be used across device types as long as the `type` and meaning remains the same. The `label` can also be used across different classes and the `type` and meaning MAY differ.

Each `uid` contains a `label` that describes the capability. To remain backwards compatibility `uid` and `label` mappings MUST never be removed or changed. A device MAY however omit the `uid` when it is not implemented or the device no longer use it. Additional capability items MAY be added with a new unique `label` and new `uid`. Because the `uid` and `label` mapping are append it MAY be staticly mapped. Multiple items within a device always have unique `label` and `uid` properties.

The `uid` are devided in ranges which are described in the table below:

Unique ID ("`uid`")	Description
`0`	Invalid uid. MUST never be used.
`1 - 99`	Range defined for general use by devices. See section devices generic.
`100 - 10000`	Range defined by the device or microservice.
`10001 - 65535`	Reserved for future use

The range `1 - 99` contains `uid` and `label` mappings which MAY be implemented by all devices. The range `100 - 10000` contains `uid` and `label` mappings which MUST be the same for all devices within the same `type`. Cached (or requested) capability items for the class `info` method MAY be ignored as all mappings are static as described in this document. Implementations MAY choose to embed these static mappings.

13. Class ‘action‘

The action class is implemented to facilitate a generic layer to device specific actions.

13.1. Methods

Name	Description
"info"	Action meta-information "uid", "type", "label"
"get"	Get action value
"set"	Set action value

13.2. Properties

Name	Description
"uid"	Action unique identifier. (See class uid mappings)
"label"	Action unique label. This member MUST be a String, MUST only contain a-z, 0-9 and '-' and has a maximal length of 50 charaters. e.g "setting-1"
"type"	Action value type. This member MUST be a String, e.g: "number". See data types
"value"	Action value. As defined by "type" property

13.3. Examples

Syntax:

- -->: Data send to Device
- <--: Data send to Client

TWS Warning Unit example actions

request information from all available actions

```
--> {
          "dinetrpc"    : 1,
          "device:uid"  : "5f5f64695f73696d756c61746f725f5f",
          "req"         : "action:info",
          "time"        : 0
    }

<-- {
          "dinetrpc"    : 1,
           "device:uid"  : "5f5f64695f73696d756c61746f725f5f",
           "rep"         : "action:info",
           "time"        : 0,
           "result"      :
           [
             {
               "uid"     : 1,
               "label"   : "silence",
               "type"    : "bool"
             },
             {
               "uid"     : 2,
               "label"   : "alarm",
               "type"    : "bool"
             }
           ]
     }
```

set action uid 1 to value true

```
--> {
          "dinetrpc"    : 1,
          "id"          : 1,
          "device:uid"  : "5f5f64695f73696d756c61746f725f5f",
          "req"         : "action:set",
          "time"        : 0,
          "params"      : {
              "uid"     : 1,
              "value"   : true
          }
     }

<-- {
          "dinetrpc"    : 1,
          "id"          : 1,
          "device:uid"  : "5f5f64695f73696d756c61746f725f5f",
          "rep"         : "action:set",
          "time"        : 0,
          "result"      : {
              "uid"     : 1,
              "value"   : true,
          }
     }
```

set invalid action uid 255 to value true

```
--> {
        "dinetrpc"    : 1,
        "id"          : 1,
        "device:uid"  : "5f5f64695f73696d756c61746f725f5f",
        "req"         : "action:set",
        "time"        : 0,
        "params"      : {
            "uid"     : 255,
            "value"   : true
        }
    }

<-- {
        "dinetrpc"    : 1,
        "id"          : 1,
        "device:uid"  : "5f5f64695f73696d756c61746f725f5f",
        "rep"         : "action:set",
        "time"        : 0,
        "error"       : {
            "code"    : 4,
            "data"    : {
                "uid"     : 255,
            }
        }
    }
```

14. Class ‘config‘

The config class is implemented to facilitate a generic interface to device specific configuration values. Configuration should always be implemented as non-volatile. A set action results in a write to persistent memory.

14.1. Methods

Name	Description
"info"	Configuration meta-information
"get"	Get single configuration value
"set"	Set single configuration value
"reset"	Reset single configuration value to default

14.2. Properties

"config:info" method properties

Name	Description
"uid"	Config unique identifier. (See class uid mappings)
"label"	Label of config (machine-readable)
"type"	Config value type. This member MUST be a String, e.g: "number". See data types
"default"	Default configuration value stored in application. See data types

```
--> {
        "dinetrpc"    : 1,
        "device:uid"  : "5f5f64695f73696d756c61746f725f5f",
        "req"         : "config:info"
    }

<-- {
        "dinetrpc"    : 1,
        "device:uid"  : "5f5f64695f73696d756c61746f725f5f",
        "rep"         : "config:info",
        "result"      :
        [
          {
            "uid"     : 1,
            "label"   : "token",
            "type"    : "number"
            "default" : 0
          }
        ]
    }
```

"config:get" method properties

Name	Description
"uid"	Config unique identifier. (See class uid mappings)

```
--> {
        "dinetrpc"    : 1,
        "id"          : 1,
        "device:uid"  : "5f5f64695f73696d756c61746f725f5f",
        "req"         : "config:get",
        "params"      : {
            "uid"     : 1
        }
    }

<-- {
        "dinetrpc"    : 1,
        "id"          : 1,
        "device:uid"  : "5f5f64695f73696d756c61746f725f5f",
        "rep"         : "config:get"
        "result"      : {
            "uid"     : 1,
            "value"   : 1337
        }
    }
```

"config:set" method properties

Name	Description
"uid"	Config unique identifier. (See class uid mappings)
"value"	Config value.

```
--> {
        "dinetrpc"    : 1,
        "id"          : 1,
        "device:uid"  : "5f5f64695f73696d756c61746f725f5f",
        "req"         : "config:set",
        "params"      : {
            "uid"     : 1,
            "value"   : 1337
        }
    }

<-- {
        "dinetrpc"    : 1,
        "id"          : 1,
        "device:uid"  : "5f5f64695f73696d756c61746f725f5f",
        "rep"         : "config:set"
    }
```

"config:reset" method properties

Name	Description
"uid"	Config unique identifier. (See class uid mappings)

```
-->  {
        "dinetrpc"    : 1,
        "id"          : 1,
        "device:uid"  : "5f5f64695f73696d756c61746f725f5f",
        "req"         : "config:reset",
        "params"      : {
            "uid"     : 1
        }
    }

<-- {
        "dinetrpc"    : 1,
        "id"          : 1,
        "device:uid"  : "5f5f64695f73696d756c61746f725f5f",
        "rep"         : "config:reset"
    }
```

15. Class ‘connection‘

15.1. Methods

Name	Description
"connect"	Published when a new connection is created
"disconnect"	Published when a connection is closed
"info"	Retreive connection information

15.2. Properties

15.2.1. ‘"connection:connect"‘ method properties

Published when a new connection is created.

Name	Description
"peer"	Remote IPv4 host and port. MUST be in format "0.0.0.0:0".

Examples Syntax:

- -->: Data send to Device
- <--: Data send to Client

publish connect originates from the device proxy (di-smp)

```
<-- {
        "dinetrpc"    : 1,
        "device:uid"  : "5f5f64695f73696d756c61746f725f5f",
        "pub"         : "connection:connect",
        "time"        : 0,
        "result"      :
        [{
            "peer"    : "127.0.0.1:1337"
        }]
    }
```

15.2.2. ‘"connection:disconnect"‘ method properties

Published when a connection is closed.

Name	Description
"peer"	Remote IPv4 host and port. MUST be in format "0.0.0.0:0".

"code"	Error code (if applicable). MUST be a Number. See error codes.

Examples Syntax:

- -->: Data send to Device
- <--: Data send to Client

publish disconnect originates from the device proxy (di-smp)

```
<-- {
        "dinetrpc"    : 1,
        "device:uid"  : "5f5f64695f73696d756c61746f725f5f",
        "pub"         : "connection:disconnect",
        "time"        : 0,
        "result"      :
        [{
            "peer"    : "127.0.0.1:1337"
        }]
    }
```

15.2.3. '"connection:info"' method properties

Retreive connection information.

Name	Type	Description
"timeout"	u32	The timeout in seconds that this device is considered offline.
"gateways"	array of device uids	The gateways that are connected to this device. If this is empty, the device itself is considered as a gateway (DNCM, Service).
"transport"	string	The transport layer of the connection, see the key transport.
key of transport	struct	Detailed information about the transport connection.

```
<-- {
        "dinetrpc"        : 1,
        "device:uid"      : "5f5f64695f73696d756c61746f725f5f",
        "pub"             : "connection:info",
        "time"            : 0,
        "result"          :
        [{
            "timeout"     : 60,
            "gateways"    : [
                 "000fc2411d0026000a47363236373834",
                 "000fc2411d0026000a47363236373835"
             ]
             "transport"   : "cellular",
             "cellular": {
                 "imsi"    : "310978123234789",
                 "iccid"   : "8991101200003204514",
                 "imei"    : "256954585566452",
                 "gprs_apn": "internet",
                 "operator": "kpn"
             }
         }]
    }
```

For gateways which are expected but are unreachable or not present a `device:uid` with all zeros is returned.

15.2.4. Transport

The transport contains one of the following values:

service

Service transport is a connection between a service and the proxy.

can

CAN transport is a connection over a Controller Area Network bus.

Name	Type	Description
`"nodeid"`	u32	The DI-CAN node id of the transport on the bus.

legacy

Legacy transport is a connection over the legacy TCP server.

Name	Type	Description
`"database_id"`	u32	The id of the record in the legacy database.
`"imsi"`	string	The International Mobile Subscriber Identity number.
`"iccid"`	string	The Integrated Circuit Card ID of the SIM chip.
`"imei"`	string	The International Mobile Equipment Identity of the phone.
`"gprs_apn"`	string	The Access Point Name of the GPRS gateway.
`"endpoint"`	string	The URI where the devices connect to.

cellular

Cellular transport is a connection over 3G/4G.

Name	Type	Description
`"operator"`	string	The mobile network operator.
`"imsi"`	string	The International Mobile Subscriber Identity number.
`"iccid"`	string	The Integrated Circuit Card ID of the SIM chip.
`"imei"`	string	The International Mobile Equipment Identity of the phone.
`"gprs_apn"`	string	The Access Point Name of the GPRS gateway.
`"endpoint"`	string	The URI where the devices connect to.

lora

LoRa transport is a connection over a LoRa network.

Name	Type	Description
`"dev_eui"`	string	The 64 bit unique identifier .
`"operator"`	string	The mobile network operator.

nbiot

NB-IoT transport is a connection over a NB-IoT network.

Name	Type	Description
`"imei"`	string	The International Mobile Equipment Identity of the modem.
`"operator"`	string	The mobile network operator.

15.2.5. Network operator

The mobile network operator contains one of the following values:

Value	Description
"kpn"	Koninklijke KPN mobile operator.
"tmobile"	The T-Mobile mobile operator.
"vodafone"	The Vodafone mobile operator.
"proximus"	Proximus Groep mobile operator.
"ttn"	The Things Network mobile operator.

16. Class ‘device‘

16.1. Methods

Class available methods:

Name	Description
"info"	Device information (static)
"data"	Device runtime information data.
"ping"	Replies empty reply (without error, or value).
"reset"	Request device hardware reset.
"errors"	Replies "errors" properties
"user:data"	Device data that is configured by the user.

16.2. Properties

16.2.1. ‘"device:info"‘ properties

Device information properties. These properties will not change during runtime and MUST always be static.

Name	Description
"type"	Device type (machine-readable). MUST be a String. See type
"version"	Hardware and Firmware versions of all boards in a device. See specific devices for detailed description

16.2.1.1 Type Devices are distinguished with the type-label. The maximum length of the string is 64 characters. The enumerator is used in the embedded MessagePack protocol.

Type	Description
""	Invalid device
"tws-3000-wum"	TWS 3000, warning unit mobile
"tws-3000-duu"	TWS 3000, detection unit ultrasonic
"tws-3000-dum"	TWS 3000, detection unit manual
"zkl-3000"	ZKL 3000
"zkl-3000-rc"	ZKL 3000 RC
"zkl-3000-rcc"	ZKL 3000 RC-C
"greenhub-3000"	GRB 3000
"crtm-gateway"	CRTM Gateway
"crtm-sensor"	CRTM Sensor
"crm-3000"	CRM 3000

"dncm"	DNCM
"reed-sensor"	Reed Sensor

16.2.1.2 State All devices have a generic state:

State	Description
"service"	Device is in service state, it can only be configured and used by the device vendor
"idle"	Device is in idle state
"armed"	Device is in armed state, the device token is configured (using config class).
"active"	Device is in active state, the device activation state is configured (using config class).

"device:data" properties

Device data properties. These properties MAY change during runtime.

Name	Description
"state"	Generic device state. MUST be a string, See device state
"error"	Device error state. MUST be a bool
"errors"	Array of raised device errors numbers. MUST be an array. When no error is raised the property MUST be omitted

16.2.2. '"device:errors"' method

When `"device:data" error` property is set to `true`. Then an array of di_error's is returned. When the `"device:data" error` property is `false`, an empty array (i.e. no result) is sent.

"device:gateways"

The device replies after request with an array of Gateway `device:uid` strings.

16.2.3. '"device:user:data"' properties

Device data that is configured by the user.

Name	Description
"serialnr"	The serial number of the device (Dual Inventive code)
"ownerlabel"	The owner label of the device (Owner code)
"name"	The name of the device. This is the ownerlabel or serialnr if ownerlabel is not set
"remark:rts"	The remark in the real time status
"gps:lat"	The GPS latitude
"gps:lon"	The GPS longitude

```
--> {
       "dinetrpc"       : 1,
       "id"             : 1,
       "device:uid"     : "12345678912345678912345678912345",
       "req"            : "device:user:data",
       "time"           : 123456789
    }

<-- {
        "dinetrpc"       : 1,
        "id"             : 1,
        "device:uid"     : "12345678912345678912345678912345",
        "rep"            : "device:user:data",
        "time"           : 234567891,
        "result"         : [{
            "serialnr"   : "T001",
            "ownerlabel" : "005.006",
            "name"       : "005.006",
            "remark:rts" : "near switch 5856",
            "gps:lat"    : 51.5859700,
            "gps:lon"    : 5.1954300
        }]
    }
```

16.3. Device Unique ID (‘device:uid‘)

A device is identified in with unique device identifier string (128-bit hex encoded number). MUST be exactly 32 ascii-characters in lowercase, e.g: `"005f64695f73696d756c61746f725f5f"`.

The first 8 bits (1 byte) are reserved for identification of device groups (not device types!). For example, legacy devices are under a different group to prevent collisions with other devices. The table below lists all available groups.

Byte	Description
`0x00`	Regular DI-Net devices
`0x01 <td> Legacy devices <tr><td>`0x02	Simulators
`0x04`	NB-IoT devices
`0x05`	LoRa devices
`0xfe`	Services (parent for other devices)
`0xff`	Artificial device data (static/mock)

The following Device UID’s are invalid:

- A device UID with all zero’s: `"00000000000000000000000000000000"`
- A device UID with characters other than `a-f` (`0x61-0x66`), `0-9` (`0x30-0x39`)
- All device UID’s with the first 8 bits not listed in the table above

16.3.1. Microservices

Microservices are also able to login with their own "Device" Unique ID. Every microservice has its own prefix as described in the table below.

Prefix	Description
`0xfe00`	NB-IoT interface
`0xfe01`	LoRa interface
`0xfe02`	CP3000 interface

17. Class ‘log‘

The log class is implemented to facilitate logging information from devices and applications. Naming of the methods is based on the UNIX syslog facility. Messages MUST be published.

17.1. Methods

Name	Description
`"crit"`	Critical message
`"err"`	Error message
`"warn"`	Warning message
`"info"`	Info message
`"debug"`	Debug message

17.2. Properties

Name	Description
`"msg"`	Human readable log message up to 160 ascii characters

17.3. Examples

Syntax:

- -->: Data send to Device
- <--: Data send to Client

publish message "Hello World!" from the device

```
<-- {
        "dinetrpc"    : 1,
        "device:uid"  : "5f5f64695f73696d756c61746f725f5f",
        "pub"         : "log:info",
        "time"        : 0,
        "result"      :
        [{
            "msg"     : "Hello World!"
        }]
    }
```

18. Class ‘message‘

The message class is implemented to facilitate SMS, Push and Email messages. It is very straightforward.

18.1. Methods

Name	Description
"sms"	Send SMS message, see SMS message
"push"	Send Push message, see Push message
"push:register"	Send Push register message, see Push register message
"email"	Send Email message, see Email message
"status"	Contains the status of the delivered message. Useful for logging purpose

18.1.1. SMS message

For sending a SMS message 2 parameters are required.

Name	Description
"dest"	String or Array of Strings which contain the destination phonenumbers
"message"	The body of the SMS message which must be sent
"sender_id"	A custom ID that contains up to 11 alphanumeric characters, including at least one letter and no spaces. The sender ID is displayed as the message sender in the sms on the receiving device. (optional)

In order to receive message:status messages the member user:id or device:uid is required. When this member is missing, no message:status message is sent.

18.1.2. Push message

Depending on the kind of push message some parameters are required.

Notification Pushes a notification in the notification tray.

Required

- dest
- title; or

- message; or
- title and message; or
- title_key; or
- title_key and title_params; or
- message_key; or
- message_key and message_params; or
- title_key and message_key; or
- title_key and title_params and message_key; or
- title_key and message_key and message_params; or
- title_key and title_params and message_key and message_params

Optional

- icon
- sound
- tag
- action
- collapsekey
- priority

Data Pushes raw data to the device. The device does nothing with the data and it is the responsibility of the programmer to do cetain actions.

Required

- dest
- data

Badge Pushes a badge change to the device.

Required

- dest
- badge

Some platforms support a combination of types (for example a push and data combined). See Messaging proxy for a detailed list of platform specific dependencies.

Name	Description
"dest"	Integer or Array of Integers which contain the destination user ids
"title"	The title of the Push message which must be sent
"message"	The body of the Push message which must be sent
"title_key"	The translation key for the title
"title_params"	The translation parameters for the title.
"message_key"	The translation key for the message
"message_params"	The translation parameters for the message.
"icon"	The notification icon in the push message
"sound"	The sound to play when the push message arrives
"tag"	The identifier to replace existing notifications
"action"	The action to perform when the user clicks on the notification
"data"	The custom data to send
"collapsekey"	The identifier to group messages. Only one message per group is send to the device
"priority"	The priority to send this push message
"badge"	The value of the badge on the home screen app icon. If not specified the badge is not changed. If set to 0 the badge is removed.
"sender_id"	A custom ID that contains up to 11 alphanumeric characters, including at least one letter and no spaces. The sender ID is displayed as the message sender in the sms on the receiving device. (optional)

The `title_params` and `message_params` is an object with all the variable values. The key represents the variable in the translation (in the format `{{.KEY_NAME}}`). The value is the value to insert into the variable. See `Go text/template` for more information about translation templates. When a variable is missing the value `<no value>` is printed.

In order to receive message:status messages the member user:id is required. When this member is missing, no message:status message is sent.

18.1.3. Push register message

For sending a Push register message 4 parameters are required.

Name	Description
"user:id"	The user id of the user to register
"platform"	The platform of the device, see mobile platforms
"token"	The token of the device
"device:uuid"	The device universally unique identifier. This value must be unique for every device independent of the mobile platform

18.1.3.1 Mobile platforms There are multiple mobile platforms to register push messaging for as described in the table below.

Name	Description
"fcm"	Android
"apns"	iOS

"wns"	UWP

18.1.4. Email message

For sending a Email message 3 parameters are required.

Name	Description
"dest"	String or Array of Strings which contain the destination email addresses
"subject"	The subject of the Email message which must be sent
"message"	The body of the Email message which must be sent

18.1.5. Status message

The status message is sent when the requested message is delivered to the endpoint. The status in the status message contains the delivery medium (Push or SMS) or if the delivery has failed.

The user:id member is used to indicate who received this message, this is however optional (not all SMS messages belongs to an MTinfo 3000 user. The device:uid member is used to indicate which device has sent this message, this is however optional (not all messages are sent from a device). Either user:id or device:uid is required.

Name	Description
"title"	The title of the message that is sent (optional)
"message"	The body of the message that is sent (mandatory)
"phone"	The telephone number that is used when it was an SMS message (mandatory for SMS)
"status"	The delivery medium that is used or failed when the message couln't be delivered, see delivery status (mandatory)

18.1.5.1 Delivery status

The delivery status is used to notify the caller about the status of the delivery. The following status values could be replied:

Name	Description
"sms"	The message is delivered by SMS
"push"	The message is delivered by Push
"failed"	The message couln't be delivered

18.2. Examples

Send SMS:

```
{
    "req": "message:sms",
    "project:id": 1,
    "id": 1,
    "dinetrpc": 1,
    "time": 1234567890,
    "params": [{
        "dest": "0629584133",
        "message": "Test SMS body 1"
    }]
}
```

Send Push (no translations):

```
{
    "req": "message:push",
    "project:id": 1,
    "id": 1,
    "dinetrpc": 1,
    "time": 1234567890,
    "params": [{
        "dest":  15,
        "title": "Push message",
         "message": "You've got a push message"
    }]
}
```

Send Push (with translations):

```
{
    "req": "message:push",
    "project:id": 1,
    "id": 1,
    "dinetrpc": 1,
    "time": 1234567890,
    "params": [{
        "dest":  15,
        "title_key": "device_title",
         "title_params": {
             "Name": "T001"
         },
         "message_key": "crtm_thresholdreached_upper",
         "message_params": {
             "Name": "T001",
             "Value": 23
         }
    }]
}
```

Send Push register:

```
{
    "req": "message:push:register",
    "project:id": 1,
    "id": 1,
    "dinetrpc": 1,
    "time": 1234567890,
    "params": {
        "user:id":  15,
        "platform": "fcm",
         "token": "123456789123456789123456789",
         "device:uuid": "device0006aabbccdd"
    }
}
```

Send Email:

```
{
    "req": "message:email",
    "project:id": 1,
    "id": 1,
    "dinetrpc": 1,
    "time": 1234567890,
    "params": [{
        "dest": "bill.gates@microsoft.com",
        "subject": "Test Email subject 1",
         "message": "Test Email body 1"
    }]
}
```

19. Class ‘monalert‘

The `monalert` class is used for user defined value monitoring and generating alert messages.

19.1. Methods

Name	Description
"create"	Create a new alert configuration. This configuration generates alerts when certain criteria is met.
"update"	Update an existing alert configuration.
"info"	Returns the alert configuration information (rules, subscribed users, devices)
"subscribe"	Subscribe on an existing alert configuration.
"unsubscribe"	Unsubscribe on an existing alert configuration. When the user is the last user in the configuration, the configuration is removed also.
"list:device"	List all the alert configurations of a specific device.
"list:user"	List all the alert configurations of a specific user.

19.2. Properties

19.2.1. ‘"monalert:create"‘ properties

Create a new alert configuration. This configuration generates alerts when certain criteria is met. The property `"user:id"` is REQUIRED which user created and receives notifications.

Request The request requires the following properties:

Name	Description
"window"	Property for creating rules over a period of time. See window.
"measurement"	The measurement that contains the data.
"select"	Property when using functions. See select. This member is REQUIRED when `"window"` is used.
"when"	Rule that triggers an alert. When the alert is triggered, no new alert is generated after the `"reset"` condition is met. See lambda expression. This member is REQUIRED.
"reset"	Rule that enables the trigger of an alert. When this condition is met and `"when"` thereafter, a new alert is send. See lambda expression. This member is REQUIRED.
"transmsgkey"	Translation message key of the alert message. See translation class message keys.
"devices"	An array of devices as `device:uid`.
"backoff"	The backoff timer in seconds to wait for sending new messages. If within the time window the reset and when is triggered. A new message is send right after the `"backoff"` window is passed.

"active"	The rules that determine when the monitoring is active. When active is equals to "null" The monitoring is always active.

Response

Name	Description
"id"	The ID of the created configuration as u64 number.

19.2.2. '"monalert:update"' properties

Update an existing alert configuration. The property "user:id" is REQUIRED to determine who wants to change the notification. Because multiple users can be subscribed to this configuration the following scenarios are executed (depending on the number of subscriptions).

Single user subscription

When there is a user subscribed, the configuration is changed and the updated configuration ID is the same as the old configuration ID.

Multiple users subscribtions

When there are multiple users subscribed, a new configuration ID is created and the existing configuration is duplicated. The user is unsubscribed from the previous configuration and subscribed to the new configuration.

Only active is changed When only the active member is changed, the member is updated and the updated configuration ID is the same as the old configuration ID.

Request The request requires the following properties:

Name	Description
"id"	The ID of the configuration to change.
"window"	Use this property when you want to create rules over a period of time. See window.
"measurement"	The measurement that contains the data.
"select"	Use this property when you want to use functions. See select. This member is REQUIRED when "window" is used.
"when"	Rule that triggers an alert. When the alert is triggered, no new alert is generated after the "reset" condition is met. See lambda expression. This member is REQUIRED.
"reset"	Rule that enables the trigger of an alert. When this condition is met and "when" thereafter, a new alert is send. See lambda expression. This member is REQUIRED.
"transmsgkey"	Translation message key of the alert message. See translation class message keys.
"devices"	An array of devices as device:uid.
"backoff"	The backoff timer in seconds to wait for sending new messages. If within the time window the reset and when is triggered. A new message is send right after the "backoff" window is passed.
"active"	The rules that determine when the monitoring is active. When active is equals to "null" The monitoring is always active.

Response

Name	Description
"id"	The ID of the created configuration as u64 number.

19.2.3. ‘"monalert:info"‘ properties

Returns the alert configuration information (rules, subscribed users, devices).

Request The request requires the following properties:

Name	Description
"id"	The ID of the created configuration as u64 number.

19.2.3.1 Response The response contains the following information:

Name	Description
"id"	The ID of the configuration as u64 number.
"window"	This property contains the period. See window. (OPTIONAL)
"measurement"	The measurement that contains the data.
"select"	This property contains additional variables with functions. See select. (OPTIONAL)
"when"	Rule that triggers an alert. When the alert is triggered, no new alert is generated after the "reset" condition is met. See lambda expression.
"reset"	Rule that enables the trigger of an alert. When this condition is met and "when" thereafter, a new alert is send. See lambda expression.
"transmsgkey"	Translation message key of the alert message. See translation class message keys.
"devices"	An array of devices as `device:uid`.
"users"	A list of users.
"backoff"	The backoff timer in seconds to wait for sending new messages. If within the time window the reset and when is triggered. A new message is send right after the "backoff" window is passed.

19.2.4. ‘"monalert:subscribe"‘ properties

Subscribe on an existing alert configuration. The property "user:id" is REQUIRED to determine which user to subscribe.

Request The request requires the following properties:

Name	Description
"id"	The ID of the created configuration as u64 number.
"active"	The rules that determine when the monitoring is active. When active is equals to "null" The monitoring is always active.

19.2.5. ‘"monalert:unsubscribe"‘ properties

Unsubscribe a user from an existing alert configuration. The property "user:id" is REQUIRED to determine which user to subscribe. When the user is the last subscriber the configuration is automaticly removed.

Request The request requires the following properties:

Name	Description
`"id"`	The ID of the created configuration as u64 number.

Response The response contains the following information:

Name	Description
`"removed"`	True when the last user was unsubscribed and the configuration is removed. False when there are remaining users.

19.2.6. ‘"monalert:list:device"‘ properties

List all the alert configurations of a specific device. The property `"device:uid"` is REQUIRED to determine for which device the list is requested.

Response An array of [monalert:info responses]().

19.2.7. ‘"monalert:list:user"‘ properties

List all the alert configurations of a specific user. The property `"user:id"` is REQUIRED to determine for which user the list is requested.

Response An array of [monalert:info responses]().

19.2.7.1 Window Window is a structure that defines the interval and period of the alert configuration.

Name	Description	Example
`"every"`	The interval in seconds to run this alert configuration.	`180`
`"period"`	The time period in seconds of samples to select.	`10`

19.2.7.2 Select Select is a structure that defines variables based on functions. The key of the structure is the new variable. The value is an array with index 0 the function and the remaining indexes the parameters of the function.

The following aggregate functions can be used with parameters (reduce to one entry in window):

Function	Description	Parameters
`"count"`	returns the number of non-null values in the window.	`"column"` or `"distinct"` function
`"distinct"`	returns the unique values in the window.	`"column"`
`"mean"`	returns the arithmetic mean (average) of values in the window.	`"column"`

"median"	returns the middle value from a sorted list of values in the window.	"column"
"mode"	returns the most frequent value in the window.	"column"
"spread"	returns the difference between the min and max of the values in the window.	"column"
"stddev"	returns the standard deviation of the values in the window.	"column"
"sum"	returns the sum of the values in the window.	"column"

The following selector functions can be used with parameters (reduce to one entry in window):

Function	Description	Parameters
"first"	returns the oldest value in the window.	"column"
"last"	returns the newest value in the window.	"column"
"max"	returns the greatest value in the window.	"column"
"min"	returns the smallest value in the window.	"column"
"percentile"	returns the Nth percentile value in the window.	"column", "N"

The following transformation functions can be used:

Function	Description	Parameters
"cumulative_sum"	returns the running total of subsequent values in the window	"column"
"derivative"	returns the rate of change between subsequent values in the window	"column", "duration"
"difference"	returns the rate of substraction between subsequent values in the window	"column"
"elapsed"	returns the difference between subsequent value timestamps in the window	"column", "duration"
"moving_average"	returns the rolling average across a window of sbsequent values in the window	"column", "N"

19.2.7.3 Lambda expression

A lambda expression contains logical, relational and arithmetic operations and results in a boolean. The operations contains a left side ("L"), operation and right side ("R").

Arithmetic operation	Description
"+"	Add finds the sum of "L" and "R".
"-"	Substract finds the difference between "L" and "R".
"*"	Multiplication finds the product of "L" and "R".
"/"	Division finds the quotient of "L" and "R".

Logical operation	Description
"and"	Returns true when "L" is true and "R" is true
"or"	Returns true when "L" is true or "R" is true

Relational operation	Description
"=="	Returns true when "L" is equal to "R"
"!="	Returns true when "L" is not equal to "R"
"\>"	Returns true when "L" is greater than "R"
"\<"	Returns true when "L" is less than "R"
"\<="	Returns true when "L" is less than or equal to "R"
"\>="	Returns true when "L" is greater than or equal to "R"

Example:

```
[
  [
    "temperature1",
    ">",
    10
  ],
  "or",
  [
    "temperature1",
    "<=",
     [
       "temperature2",
       "+",
       "2"
     ]
  ]
]
```

This expression is the same as "temperature1 \> 10 || temperature1 \<= (temperature2 + 2)".

19.2.7.4 Measurement Measurement is the "table" that contains the data. Currently the following values are valid:

Name	Description
"event"	Messages (pub/rep) sent from the device to the server
"device"	The error and state of the device.
"device_error"	The errors of the device.
"sensor"	The sensor data of the device.
"notify"	The notify data of the device.

19.2.7.5 User User contains all the information about the user that is subscribed to a configuration alert.

Name	Description
"id"	The user id as u32.
"active"	The rules that determine when the monitoring is active. When active is equals to "null" The monitoring is always active.

19.2.7.6 Active Active describes when the monitoring starts and stops. All values are REQUIRED.

Name	Description
"start"	The time in seconds from midnight when the monitoring activates.
"stop"	The time in seconds from midnight when the monitoring deactives.
"time_zone"	The time zone that is used to determinate midnight. The time zone is a value from the IANA time zone database such as "America/New_York". See list of tz database time zones for a list of valid time zones.

20. Class ‘notify‘

The notify class is used to report state changes.

20.1. Methods

Class available methods:

Name	Description
"info"	Information of available notifications
"data"	Notification data

20.2. Properties

"notify:info" method properties

Name	Description
"uid"	Notification unique identifier. (See class uid mappings)
"label"	Notification unique label. This member MUST be a String, MUST only contain a-z, 0-9 and ’-’ and has a maximal length of 50 charaters. e.g "battery-level"
"type"	Notification value type. This member MUST be a String, e.g: "number". See data types

"notify:data" properties

Name	Description
"uid"	Notification unique identifier. (See class uid mappings)
"time"	Notification DI-Net time.
"value"	Notification value. Actual value is formated according to "type" property (as reported by "notify:info" method).

21. Class ‘project‘

The project class is used for project management from an API or frontend. It groups multiple devices into an object for execution by a business logic.

21.1. Methods

Name	Description

`"create"`	Submit a project setup and create a new project
`"setup"`	Submit a new project setup
`"planning:{submethod}"`	Request planning action, see submethod
`"design:{submethod}"`	Request design action, see submethod
`"release"`	Request for project release
`"return"`	Request for project return
`"unlock"`	Request for project unlock
`"lock"`	Request for project lock
`"activate"`	Request for project activation
`"deactivate"`	Request for project deactivation
`"counter"`	TWS 3000 project train counters, see Counter
`"status"`	Project (safety) status, see Status
`"list"`	Request a list of (running) projects, see List

21.2. Submethods

Name	Description
`"concept"`	Submit a new concept
`"ready"`	Change the state to ready for verification
`"verify"`	Verify the planning or design
`"validate"`	Validate the planning or design
`"reject"`	Reject a planning or design (when validation or verification is rejected)

21.3. Properties

Name	Description
`"id"`	Project unique ID. This member MUST be a Number without fractional part. Defined by the front-end
`"project"`	Project data, this MUST be an object and is REQUIRED for all requests.
`"devices"`	List of devices selected on `"unlock"`. This member MUST be an Array and is REQUIRED on `"unlock"`.
`"groups"`	List of groups valid for `"activate"` and `"deactivate"`. This member MUST be an Array and is REQUIRED for `"activate"` and `"deactivate"`.

21.3.1. ‘project‘ property

Property name	Description
`"setup"`	The project setup. This member is REQUIRED.
`"planning"`	The project planning. This member is REQUIRED.
`"design"`	The project design. This member is REQUIRED.

21.3.1.1 ‘setup‘ property The setup property is used for information in the project "setup" step.

Name	Description
`"project:id"`	The project identifier, the same as in the message header. This member is OPTIONAL, since it is redundant and ignored by the backend and secure server.
`"type"`	The project type, a string which is REQUIRED. Only `"tws"`, `"rc"` or `"crtm"` are valid.
`"roles"`	An object which contains a structure of project roles with user-lists of user ID's assigned to each role. This member is not used for project type `"crtm"`

21.3.1.2 'planning' property The planning property is used for information in the project "planning" step.

Name	Description
`"groups"`	List of Planning groups
`"devices"`	List of devices
`"wa-selection"`	The wa alarm for wum devices during the error or detection state. See wu-wa-selection enum for possible values. REQUIRED and used only for `"tws"`.

21.3.1.3 Planning groups

Name	Description
`"id"`	The group identifier, REQUIRED, MU unique and a Number.
`"name"`	The group name, OPTIONAL, MUS String.
`"operators"`	List of users, REQUIRED.

21.3.1.4 Design The project design property is used for information in the project "design" step.

Name	Description
`"devices"`	List of devices

21.3.1.5 Groups

Name	Description
`"id"`	The group identifier, REQUIRED, MUST be unique and a Number.

21.3.2. Users

The users are contained in a project to check if a valid user executes an action.

Name	Description
`"user:id"`	User unique ID, this member is REQUIRED.
`"from"`	DI-Net timestamp from which UTC time this user ID is allowed. This value is OPTIONAL. When the value is `0`, `null` or not available, it is interpreted as invalid.
`"to"`	DI-Net timestamp to which UTC time this user ID is allowed. This value is OPTIONAL. When the value is `0`, `null` or not available, it is interpreted as invalid.

21.3.3. Devices

The devices are contained in a project with different parameters.

Name	Description
"device:uid"	Device unique ID, see device:uid (REQUIRED)
"group:id"	Group identifier of device. This member MUST be an Number and is REQUIRED for devices which are part of the project's design.
"role"	The role of a device on the project. This member MUST be an string and contain "strike_in" or "strike_out". This member is REQUIRED for devices which part of a "tws"-project's design.
"option"	Device configuration of part of the project design. Used for "crtm" projects. See CRTM business logic for an example.

21.3.4. 'counter' method

The project counter messages are the train-counters published by the backend on change.

Name	Description
"id"	Channel identifier, NULL for global counter
"value"	Counter value

21.3.5. 'status' method

The `project:status` method properties are the aggregated values of all possible parameters which influence the safety of the project workzone.

Name	Description
"traincounter"	Global train counter value (Number)
"system_error"	Boolean whether a system error is occurred
"device_error"	Boolean whether at least one device has an error (or not)
"safe"	Whether the project is safe ("traincounter" is non-zero or "system_error")

21.3.6. 'state' method

The project state messages are published by the backend when the status of the project changes.

Name	Description
"status"	The status of the project. The current status values are: "ready", "released", "unlocked" and "activated"

21.3.7. 'list' method

The `project:list` method returns a list of (running) projects result from the secure server. Only projects that are released, unlocked or activated are returned.

Request

Name	Description
"type"	The type of project to retrieve. Only "tws", "rc" or "crtm" are valid. When the type is empty, all types are returned.

Reply

Name	Description
`"project:id"`	The id of the project
`"project:status"`	The status of the project. The current status are `"released"`, `"unlocked"` and `"activated"`
`"project:data"`	The current project data
`"unlocked_devices"`	List of devices that are unlocked. Or an empty array when the status is not `"unlocked"` or `"activated"`
`"active_groups"`	List of groups that are currently active. Or an empty array when the status is not `"activated"`

21.4. Examples

- See Secure Server

22. Class ‘realtime‘

The `realtime` class is used to facilitate real-time status information from the backend to the frontend and is solely used by the Websocket server.

22.1. Methods

Name	Description
`"request"`	Realtime status request
`"data"`	Realtime status data-message

22.1.1. ‘request‘ method

The `realtime:request` calls originate from the websocket client (e.g javascript) and are routed to the MTIWSS http endpoint for authentication and realtime data field subscriptions.

22.1.2. ‘data‘ method

When new device realtime data arrives it is pushed over the websocket to the connected client as a `realtime:data` message.

22.2. Frontend JSON API

The PHP web-interface MUST provide a JSON API to provide data to the back-end from information stored in the front-end database. The API information is requested by the Websocket server.

22.2.1. WebSocket Service

For real-time status the front-end uses websockets, the WebSocket service is provided and maintained by the backend. All incoming data from the client is forwarded to the PHP JSON API. the response is read and parsed, afterwards the real-time status datasource for the particular connection is updated.

22.2.1.1 API Response The response from the PHP API is parsed by the WebSocket Service and must have the following format, otherwise the response is regarded invalid and forwarded to the client instead of handled internally.

```
<-- {
         "dinetrpc"    : 1,
         "rep"         : "realtime:request",
         "id"          : 0,
         "time"        : 1433087730341,
         "result"      :
         [
             {
                 "device:uid"  : "5f5f64695f73696d756c61746f725f5f",
                  "fields"      : [ "last_update", "device:info", "sensor:5:data" ... ]
              },
              {
                  "device:uid"  : "623",
                  "fields"      : [ "last_update", "device:info", "sensor:5:data" ... ]
              },
              {
                  "project:id"  : 12,
                  "fields"      : [ "last_update", "project:counter", ... ]
              }
          ]
      }
```

The fields property is an array of string which are the Redis keys required for the devices' or project's real-time status.

22.2.2. Real time data

When a new update is ready to be pushed from the WebSocket service, all fields are retrieved from the cache and stored in a JSON object which is then distributed to all clients which are subscribed to the particular device or project.

The final update message has the following format for devices:

```
<-- {
        "dinetrpc"    : 1,
        "pub"         : "realtime:data",
        "device:uid"  : "5f5f64695f73696d756c61746f725f5f",
        "time"        : 1433087730341,
        "result"      :
        {
            "last_update"    : "1433095090895",
            "device:info"    : {"type":"ZKL 3000 RC","last_update":1433087730341},
             "sensor:5:data"  : {"uid":5,"time":1433087730332,"value":0.0,"last_update":1433087730372},
             ...
        }
    }
```

And for projects:

```
<-- {
        "dinetrpc"    : 1,
        "pub"         : "realtime:data",
        "project:id"  : 12,
        "time"        : 1433087730341,
        "result"      :
        {
            "last_update"     : "1433095090895",
            "project:counter" : {"id":null, "value":1,"last_update":1433087730341,"time":1433087730341}
             ...
        }
    }
```

23. Class 'sensor'

The `sensor` class is used for sensing.

23.1. Methods

Name	Description
`"data"`	Retreive selected or all available sensor(s) data properties: `"uid"`, `"time"`, `"value"`.
`"info"`	Retreive selected or all available sensor(s) metadata properties: `"label"`, and `"type"`

23.2. Properties

Name	Description
"uid"	Sensor unique identifier. (See class uid mappings)
"label"	Sensor unique label. This member MUST be a String, MUST only contain a-z, 0-9 and '-' and has a maximal length of 50 charaters. e.g "bat1-percent"
"type"	Sensor value type. This member MUST be a String, e.g: "number". See data types
"time"	Sensor value absolute timestamp using "dinet:time" data type.
"value"	Sensor value. Actual value is formated according to "type" property. Mutually exclusive with "values" property.
"values"	Sensor values. See Values properties. Mutually exclusive with "value" property.

23.3. Values properties

Name	Type	Description
"interval"	Number (u64)	The interval of the sample values in nanoseconds.
"samples"	Array	The actual array of sample values. E.g "numbers"

The samples are ordered from first to last (0...N). Sample `0` starts at absolute time `"time"` and sample N ends at absolute time value `time + (N * "interval")`.

The samples value `"type"` supported is `"numbers"` (See data types).

23.4. Examples

Syntax:

- -->: Data send to Device
- <--: Data send to Client

rpc request sensor::info, uid all

```
--> {
        "dinetrpc"    : 1,
        "id"          : 1,
        "device:uid"  : "5f5f64695f73696d756c61746f725f5f",
        "req"         : "sensor:info",
        "time"        : 0
    }

<-- {
        "dinetrpc"    : 1,
        "id"          : 1,
        "device:uid"  : "5f5f64695f73696d756c61746f725f5f",
        "rep"         : "sensor:info",
        "time"        : 0,
        "result"      :
        [
          {
            "uid"     : 1,
            "label"   : "bat:1:stat",
            "type"    : "percent"
          },
          {
            "uid"     : 2,
            "label"   : "bat:1:info",
            "type"    : "string"
          },
          {
            "uid"     : 3,
            "label"   : "bat:2:stat",
            "type"    : "percent"
          },
          {
            "uid"     : 4,
            "label"   : "bat:2:info",
            "type"    : "string"
          },
        ]
    }
```

rpc request sensor:info, uid 1

```
--> {
        "dinetrpc"    : 1,
        "id"          : 1,
        "device:uid"  : "5f5f64695f73696d756c61746f725f5f",
        "req"         : "sensor:info",
        "time"        : 0,
        "params"      : { "uid" : 1 }
    }

<-- {
        "dinetrpc"    : 1,
        "id"          : 1,
        "device:uid"  : "5f5f64695f73696d756c61746f725f5f",
        "rep"         : "sensor:info",
        "time"        : 0,
        "result"      :
        [{
            "uid"     : 1,
            "label"   : "bat1-status",
            "type"    : "percent"
        }]
    }
```

publish object of sensor 1 (uid 1, battery 1 status)

```
<-- {
        "dinetrpc"    : 1,
        "device:uid"  : "5f5f64695f73696d756c61746f725f5f",
        "pub"         : "sensor:data",
        "time"        : 0,
        "result"      :
        [{
            "uid"     : 1,
            "time"    : 0,
            "value"   : 66.66
        }]
    }
```

request sensor uid 13 info, which is of enumerated type switch with 3 states

```
--> {
        "dinetrpc"    : 1,
        "id"          : 1,
        "device:uid"  : "5f5f64695f73696d756c61746f725f5f",
        "req"         : "sensor:info",
        "time"        : 0,
        "params"      : {"uid" : 13 }
    }

<-- {
        "dinetrpc"    : 1,
        "id"          : 1,
        "device:uid"  : "5f5f64695f73696d756c61746f725f5f",
        "rep"         : "sensor:info",
        "time"        : 0,
        "result"      :
        [{
            "uid": 13,
            "label": "key-switch",
            "descr": "Keyswitch state",
            "type" : "enum",
            "enum" : {
              "ON":   1,
              "OFF":  2,
              "OPER": 3
            }
        }]
    }
```

multiple values for sensor uid 100 info request

```
--> {
        // Rest of required header properties are not specified in this example
        "req"         : "sensor:info",
    }

<-- {
        // Rest of required header properties are not specified in this example
        "rep"         : "sensor:info",
        "result"      :
         [
           {
             "uid"     : 100,
             "label"   : "svs-vibration",
             "type"    : "numbers"
           },
     }
```

multiple values for sensor uid 100 data publish

```
<-- {
        // Rest of required header properties are not specified in this example
        "pub"         : "sensor:data",
        "time"        : 1535546718001,
        "result"      :
        [{
            "uid"     : 100,
            "time"    : 1535546718115,
            "values"  : {
                 "interval" : 31250,
                 "samples"  : [
                    1, 2, 3, 4
                 ]
             }
         }]
     }
```

24. Class ‘translate‘

The `translate` class is used for translated message templates send to the customer.

Methods

Class available methods:

Name	Description
"get"	Retrieve a translation

24.1. Properties

24.1.1. ‘"translate:get"‘ properties

Retrieve a translation.

Name	Description
"key"	The key of the translation (REQUIRED)
"i18n"	The language to translate to (REQUIRED)
"value"	The translated value

```
--> {
        "dinetrpc"   : 1,
        "id"         : 1,
        "project:id" : 5,
        "req"        : "translate:get",
        "time"       : 123456789,
        "params"     : {
            "key"    : "crtm_thresholdreached_upper",
            "i18n"   : "en_US"
         }
     }

<-- {
         "dinetrpc"   : 1,
         "id"         : 1,
         "project:id" : 5,
         "rep"        : "translate:get",
         "time"       : 234567891,
         "result"     : [{
             "key"    : "crtm_thresholdreached_upper",
             "i18n"   : "en_US",
             "value"  : "CRTM 3000 Sensor {{.Name}} * Temperature of this sensor has reached the {{.Value}}
     upper threshold."
         }]
     }
```

24.2. Translation message keys

The following translations message keys are defined:

Key	Description
"crtm_batterystate_crit"	The battery of the CRTM 3000 is critical
"crtm_batterystate_empty"	The battery of the CRTM 3000 is empty
"crtm_railcontactfailure"	The CRTM 3000 is not attached to the rail
"crtm_thresholdreached_lower"	The threshold of the CRTM 3000 has reached its upper threshold
"crtm_thresholdreached_upper"	The threshold of the CRTM 3000 has reached its lower threshold
"zkl_raildetectionfailure"	The detection of the ZKL 3000 failed
"zkl_batterystate_removed_empty"	The battery of the ZKL 3000: backup: removed, extern: empty
"zkl_batterystate_removed_crit"	The battery of the ZKL 3000: backup: removed, extern: critical
"zkl_batterystate_empty_removed"	The battery of the ZKL 3000: backup: empty, extern: removed
"zkl_batterystate_empty_empty"	The battery of the ZKL 3000: backup: empty, extern: empty
"zkl_batterystate_empty_crit"	The battery of the ZKL 3000: backup: empty, extern: critical
"zkl_batterystate_empty_low"	The battery of the ZKL 3000: backup: empty, extern: low
"zkl_batterystate_empty_half"	The battery of the ZKL 3000: backup: empty, extern: half full
"zkl_batterystate_empty_full"	The battery of the ZKL 3000: backup: empty, extern: full
"zkl_batterystate_empty_unknown"	The battery of the ZKL 3000: backup: empty, extern: unknown
"zkl_batterystate_crit_removed"	The battery of the ZKL 3000: backup: critical, extern: removed
"zkl_batterystate_crit_empty"	The battery of the ZKL 3000: backup: critical, extern: empty
"zkl_batterystate_crit_crit"	The battery of the ZKL 3000: backup: critical, extern: critical
"zkl_batterystate_crit_low"	The battery of the ZKL 3000: backup: critical, extern: low
"zkl_batterystate_crit_half"	The battery of the ZKL 3000: backup: critical, extern: half full
"zkl_batterystate_crit_full"	The battery of the ZKL 3000: backup: critical, extern: full
"zkl_batterystate_crit_unknown"	The battery of the ZKL 3000: backup: critical, extern: unknown

"zkl_batterystate_low_empty"	The battery of the ZKL 3000: backup: low, extern: empty
"zkl_batterystate_low_crit"	The battery of the ZKL 3000: backup: low, extern: critical
"zkl_batterystate_half_empty"	The battery of the ZKL 3000: backup: half full, extern: empty
"zkl_batterystate_half_crit"	The battery of the ZKL 3000: backup: half full, extern: critical
"zkl_batterystate_full_empty"	The battery of the ZKL 3000: backup: full, extern: empty
"zkl_batterystate_full_crit"	The battery of the ZKL 3000: backup: full, extern: critical
"zkl_batterystate_unknown_empty"	The battery of the ZKL 3000: backup: unknown, extern: empty
"zkl_batterystate_unknown_crit"	The battery of the ZKL 3000: backup: unknown, extern: critical

Variables

Global (business logic)

The following variables are available for every translation:

Name	Description
".Name"	The name of the device e.g. "T001" or "005.006"
".RTSremark"	The data of the RTS remark

Thresholds

For "crtm_thresholdreached_lower" and "crtm_thresholdreached_upper" the following variables are also available:

Name	Description
".Value"	The value of the threshold that is reached

25. Class ‘user‘

The user class is used for customer user profile data (e.g phone number, translation region).

Methods

Class available methods:

Name	Description
"data"	User runtime information data.

25.1. Properties

25.1.1. ‘"user:data"‘ properties

User data properties. These properties MAY change during runtime.

Name	Description
"phone:alarm"	Users alarm phone number
"i18n"	Users prefered language (I18N)

```
--> {
        "dinetrpc"          : 1,
        "id"                : 1,
        "user:id"           : 5,
        "req"               : "user:data",
        "time"              : 123456789
```

```
    }

<-- {
        "dinetrpc"       : 1,
        "id"             : 1,
        "user:id"        : 5,
        "rep"            : "user:data",
        "time"           : 234567891,
        "result"         : [{
            "phone:alarm" : "0123456789",
            "i18n"        : "en_US"
        }]
    }
```

26. Devices

This chapter contains the generic class item unique id mappings in the `"uid": 1-99` range.

26.1. Class ‘sensor‘

The available sensor info and data for all devices is listed below:

Unique ID	Label	Type	Description
1	bat1-voltage	number (float)	Battery #1 voltage
2	bat1-state	enum	Battery #1 state (see bat-state enumeration)
3	bat2-voltage	number (float)	Battery #2 voltage
4	bat2-state	enum	Battery #2 state (see bat-state enumeration)
5	bat3-voltage	number (float)	Battery #3 voltage
6	bat3-state	enum	Battery #3 state (see bat-state enumeration)
7	bat4-voltage	number (float)	Battery #4 voltage
8	bat4-state	enum	Battery #4 state (see bat-state enumeration)
9	charger1-voltage	number (float)	Battery charger #1 voltage
10	charger1-state	enum	Charger #1 state (see charger-state enumeration)
11	charger2-voltage	number (float)	Battery charger #2 voltage
12	charger2-state	enum	Charger #2 state (see charger-state enumeration)
13	gps	gps	GPS location (e.g from DNCM 3G modem)
14	rssi	number	Receive Strength Signal Indicator (e.g DNCM 3G modem)
15	ber	number	Bit Error Rate of wireless connection
16-99	-	-	Reserved for future use

26.2. Class ‘action‘

The available action info and data for all devices is listed below:

Unique ID	Label	Type	Description
1-99	-	-	Reserved for future use

26.3. Class ‘config‘

The available config info and data for all devices is listed below:

Unique ID	Label	Type	Description
1	token	number (u32)	See token item
2	activate	bool	See activate item
3	service	bool	See service item
4-99	-	-	Reserved for future use

26.3.1. ‘token‘ item

The token configuration item is used for locking the device to a project and control the device state between `idle` and `armed`. The token is written with a `config:set` call. And the token is removed with a `config:reset` call. Writing a token with value `0` is not allowed as this is used as the reset value when the device is in the `idle` state.

26.3.2. ‘service‘ item

The service configuration item is used to control the device state between `service` and `idle` (see `device:state`).

In `service` state the device can only be controlled and used by the vendor and the customer is unable to use devices in this state.

When in this state the device sends a heartbeat every 60 seconds.

26.3.3. ‘activate‘ item

The activate configuration item is used to control the device state between `armed` and `active` (see `device:state`).

In `idle` and `armed` state the device sensors measuring, heartbeating uses a long period (e.g 60 seconds). Errors, notifications, etcetra MAY be reported. In this states the device MAY put some functionality in sleep for power saving.

For the `active` device state a short period is used (e.g 1 second). In this state the device SHOULD execute all functionality.

26.4. Class ‘notify‘

The available notify info and data for all devices is listed below:

Unique ID	Label	Type	Description
1-99	-	-	Reserved for future use

Enumerations {##dinet_rpc_devices_generic_enum}

26.4.1. ‘bat-state‘ enumeration

Battery state enumerator as defined in the table below:

Enumerator	Value	Description
"removed"	0	Removed (no battery connected)
"empty"	1	Empty < 3%
"crit"	2	Critical > 3% <= 15%
"low"	3	Low > 15% <= 33%
"half"	4	Half full > 33 % <= 66%
"full"	5	Battery full > 66%
"unknown"	255	Unknown/unset (reserved and must never be send from a device)

26.4.2. ‘charger-state‘ enumeration

Battery charger state enumerator as defined in the table below:

Enumerator	Value	Description
"disconnected"	0	Charger is disconnected
"connected"	1	Charger is connected
"charging"	2	Battery charging in progress
"error"	3	Charger voltage is to low
"unknown"	255	Unknown/unset (reserved and must never be send from a device)

27. Device CRM 3000

27.1. Device

27.1.1. Version property

Key	Description
`"fw-crm"`	CRM firmware version
`"hw-crm"`	CRM hardware PCB version

27.2. Sensor

The available sensor info and data for the CRM 3000 is listed in the table below:

Unique ID	Label	Type	Description
1	bat1-voltage	-	See section Device Generic : Sensor
2	bat1-state	-	See section Device Generic : Sensor
100	temperature1	Number (degree celsius)	External PT1000 temperature sensor
101	temperature2	Number (degree celsius)	Internal CPU temperature sensor
102	acceleration	Structure	Acceleration data, see Acceleration data
103	cap-touch	Number	Capacitive touch sensor (Dimensionless counts (lower equals more capacity))

27.2.1. Acceleration data

The CRM 3000 acceleration sensor is represented by 3 axis

Property	Description
"x"	X-axis acceleration in mG
"y"	Y-axis acceleration in mG
"z"	Z-axis acceleration in mG

27.3. Notify

None

27.4. Config

None

28. Device CRTM Gateway

28.1. Device

28.1.1. Version property

Key	Description
`"hw-gateway"`	Gateway hardware version
`"fw-gateway"`	Gateway firmware version

28.2. Sensor

The available sensor info and data for the CRTM Gateway is listed in the table below:

Unique ID	Label	Type	Description
1	bat1-voltage	-	See section Device Generic : Sensor
2	bat1-state	-	See section Device Generic : Sensor
14	rssi	Number	RSSI of 2G/3G Modem
15	ber	Number	Bit Error Rate of wireless connection
100	temperature1	Number	Onboard temperature sensor

The CRTM Gateway publishes sensor info after connection.

28.3. Config

The available configuration info and data for the CRTM Gateway is listed in the table below:

Unique ID	Label	Type	Description
3	service	-	See Chapter Device Generic : Config
102	endpoint	string	TCP endpoint location formatted as host:port (e.g `"di-tcp.dualinventive.com:4020"`). Only Legacy devices support this.

29. Device CRTM Sensor

29.1. Device

29.1.1. Version property

Key	Description
"hw-crtm"	CRTM Sensor hardware version
"fw-crtm"	CRTM Sensor firmware version

29.2. Sensor

The available sensor info and data for the CRTM Sensor is listed in the table below:

Unique ID	Label	Type	Description
1	bat1-voltage	-	See section Device Generic : Sensor
2	bat1-state	-	See section Device Generic : Sensor
13	gps	gps	GPS coordinates of the CRTM-sensor (optional available field, depending on the source of the device)
14	rssi	Number	RSSI of wireless connection
15	ber	Number	Bit Error Rate of wireless connection
100	temperature1	Number	First temperature sensor
101	temperature2	Number	Second temperature sensor
102	rail-contact	Boolean	Actual rail contact sensor
103	rail-contact-sleep	Boolean	Rail contact sensor during sleep
104	acceleration	struct	Acceleration data, see Acceleration data

GPS coordinates can be present, when (for example) a crtm-sensor connects via the KPN LoRa network, triangular location data is given.

29.2.1. Acceleration data

The CRTM 3000 acceleration sensor is represented by 3 axis

Property	Description
"x"	X-axis acceleration in mG
"y"	Y-axis acceleration in mG
"z"	Z-axis acceleration in mG

29.3. Notify

None

29.4. Config

The available configuration info and data for the CRTM Sensor is listed in the table below:

Unique ID	Label	Type	Description
3	service	-	See Chapter Device Generic : Config
100	configuration	struct	Only for LoRa devices. See configuration
101	calibration	bool	Only for LoRa devices. 'True' start capacitive touch calibration. 'False' does nothing.
102	endpoint	string	TCP endpoint location formatted as host:port (e.g "di-tcp.dualinventive.com:4020"). Only Legacy devices support this.

29.4.1. LoRa sensor configuration

Note: Config:get retrieves the configuration from cache and is not real-time due the fact that lora modules don't support direct req/rep.

```
--> {
        "dinetrpc"    : 1,
        "id"          : 1,
        "device:uid"  : "5f5f64695f73696d756c61746f725f5f",
        "req"         : "config:set",
        "params"      : {
            "uid"     : 100,
            "value"   : {
                               "measurement_interval"       : 5,
                                "transmission_interval"      : 12,
                                "hysteresis_threshold"       : 2,
                                "num_temperatures"           : 6,
                                "measurement_report_interval": 6,
                                "num_retries"                : 1
                            }
        }
    }

<-- {
        "dinetrpc"    : 1,
        "id"          : 1,
        "device:uid"  : "5f5f64695f73696d756c61746f725f5f",
        "rep"         : "config:set"
    }
```

Configuration

CRTM LoRa sensor has the following configuration options:

Key	Min value	Default	Max value	Description
"measurement_interval"	1	5	240	Time between measurements (in minutes). After each measurement the hysteresis is checked. If the hysteresis is reached a message is sent immediately.

"transmission_↩ interval"	1	12	255	The number of measurements after it sends the data (when the hysteresis is not met).
"hysteresis_↩ threshold"	0.1	2	25.5	When the temperature difference between the measurements is greater than the threshold, a message is sent immediately.
"num_↩ temperatures"	2	6	16	The number of temperatures in a payload message. This needs to be a multiple of 2.
"measurement↩ _report_interval"	1	6	255	The number of measurements before it is added to the payload. `Time between temperatures in payload = report interval * measurement interval.`
"num_retries"	0	1	2	The number of retries to send the message before giving up. This is exclusive to the first send action.

With the default configuration the following conditions are true:

1. In the worst-case scenario every 5 min a message is sent (`measurement_interval = 5`);
2. In the best-case scenario every 1 hour a message is sent (`transmission_interval * measurement↩ _interval`);
3. When the message is lost during transmission, one retry is sent (`num_retries = 1`);
4. When the temperature after 5 minutes changes more than 2 degrees celcius, a message is sent immediately (`hysteresis_threshold`);
5. There are always 6 temperatures in a message (`num_temperatures`). 2 temperatures are new (`transmission_interval / measurement_report_interval`), the rest are previous measurements in case the message is lost during transmission.

When a key is not set, the default value is used instead. So an empty struct will reset the device to the default configurations.

30. Device DNCM

The DNCM device functions as a transparent DI-Net RPC proxy between the CAN-bus and a DI-Net RPC Lowlevel TCP data connection.

30.1. Device

30.1.1. Version property

Key	Description
`"fw-dncm"`	DNCM firmware version
`"hw-dncm"`	DNCM hardware PCB version
`"fw-modem"`	DNCM modem firmware version

30.2. Config

The available config info and data for the DNCM is listed in the table below:

Unique ID	Label	Type	Description
1-99	-	-	Reserved for Device Generic configuration items
100	dncm-tcp-host	string	Deprecated, do not use! TCP server hostname (e.g `"di-tcp.↩ dualinventive.com"`)
101	dncm-tcp-port	string	Deprecated, do not use! TCP server port (e.g `"4020"`)
102	endpoint	string	TCP endpoint location formatted as host:port (e.g `"di-tcp.↩ dualinventive.com↩ :4020"`)

30.3. Sensor

The available sensor info and data for the DNCM is listed in the table below:

Unique ID	Label	Type	Description
13	gps	gps	Modem GPS information
14	rssi	number	Modem carrier RSSI (in dBm)
15	ber	number	Bit Error Rate of wireless connection
100	dncm-temp	number	Onboard temperature sensor (in ℃)

31. Device GRB 3000

31.1. Device

31.1.1. Version property

Key	Description
"hw-grb"	GRB 3000 hardware version
"fw-grb"	GRB 3000 firmware version

31.2. Sensor

The available sensor info and data for the GRB 3000 is listed in the table below:

Unique ID	Label	Type	Description
1	bat1-voltage	-	See section Device Generic : Sensor
2	bat1-state	-	See section Device Generic : Sensor
14	rssi	number	RSSI of 2G/3G Modem
15	ber	Number	BER of Modem
100	temperature1	Number	Onboard temperature sensor

The GRB 3000 publishes sensor info after connection.

31.3. Notify

None

31.4. Config

The available configuration info and data for the GRB 3000 is listed in the table below:

Unique ID	Label	Type	Description
3	service	-	See Chapter Device Generic : Config
102	endpoint	string	TCP endpoint location formatted as host:port (e.g "di-tcp.↩ dualinventive.com↩ :4020"). Only Legacy devices support this.

32. Device Reed Sensor

32.1. Sensor

The available sensor info and data for the Reed Sensor is listed in the table below:

Unique ID	Label	Type	Description
1	bat1-voltage	-	See section Device Generic : Sensor
2	bat1-state	-	See section Device Generic : Sensor
13	gps	gps	GPS coordinates of the Reed-sensor (optional available field, depending on the source of the device)

14	rssi	Number	RSSI of wireless signal
100	temperature1	Number	First temperature sensor
102	contact-closed	Boolean	Whether the contact is closed or not

GPS coordinates can be present, when (for example) a reed-sensor connects via the KPN LoRa network, triangular location data is given.

32.2. Notify

None

32.3. Config

None

33. Device detection unit manual (DUM)

33.1. Device

33.1.1. Version property

Key	Description
"hw-dum"	DUM hardware PCB version
"fw-dum"	DUM MCU firmware version

33.2. Action

The available action info and data for the DUM is listed in the table below:

Unique ID	Label	Type	Description
1-99	-	-	See Section Device Generic : Action
100	du-train-counter	number	Number of trains in the channel
101	du-strike-role	enum (see du-strike-role	The strike role of the DUM

33.2.1. du-strike-role

Detection unit strike role values:

Enumerator	Value	Description
"unknown"	0	The device has an unknown role
"strike_in"	1	The device detect incoming trains
"strike_out"	2	The device detect outgoing trains

33.3. Sensor

The available sensor info and data for the DUM is listed in the table below:

Unique ID	Label	Type	Description
1	bat1-voltage	number	See section Device Generic : Sensor

2	bat1-state	enum	Battery #1 state (see bat-state enumeration)
9	charger1-voltage	number	See section Device Generic : Sensor
10	charger1-state	enum	Charger #1 state (see charger-state enumeration)
13	gps	gps	GPS location of 2G/3G Modem
14	rssi	number	RSSI of 2G/3G Modem
15	ber	number	Bit Error Rate of wireless connection
100	du-manual	enum du-manual	Button press detection. Deprecated use uid 102
101	-	-	Reserved for future use
102	du-counter	number (u32)	Button press detection incremental counter

The DUM sensor data publish

```
<-- {
        "dinetrpc"    : 1,
        "device:uid"  : "5f5f64695f73696d756c61746f725f5f",
        "pub"         : "sensor:data",
        "time"        : 1444637483000,
        "result"      : [
            {
                "uid"   : 1,
                "time"  : 1444637481281,
                 "value" : 12.048415652103751
             },
             {
                 "uid"   : 2,
                 "time"  : 1444637480234,
                 "value" : 4
             }
        ]
    }
```

33.3.1. du-manual detection state enum

Detection unit manual state enumerator values:

Enumerator	Value	Description
"err"	-1	Button read error
"none"	0	No button pressed detection
"both"	1	Both buttons pressed detection

33.4. Config

The available configuration info and data is listed in the table below:

Unique ID	Label	Type	Description
1	token	-	See device generic token
2	activate	-	See device generic activate
3	service	-	See device generic service

33.5. Notify

The available notify info and data for the DUM is listed in the table below:

Unique ID	Label	Type	Description
1-99	-	-	See Chapter Device Generic : Notify
100	du-detection	bool	Detection unit object detected

34. Device detection unit ultrasonic (DUU)

34.1. Device

34.1.1. Version property

Key	Description
"hw-duu"	DUU hardware PCB version
"fw-duu"	DUU MCU firmware version

34.2. Action

The available action info and data for the DUU is listed in the table below:

Unique ID	Label	Type	Description
1-99	-	-	See Section Device Generic : Action
100	du-train-counter	number	Number of trains in the channel
101	du-strike-role	enum (see du-strike-role	The strike role of the DUU

34.2.1. du-strike-role

Detection unit strike role values:

Enumerator	Value	Description
"unknown"	0	The device has an unknown role
"strike_in"	1	The device detect incoming trains
"strike_out"	2	The device detect outgoing trains

34.3. Sensor

The available sensor info and data for the DUU is listed in the table below:

Unique ID	Label	Type	Description
1	bat1-voltage	-	See section Device Generic : Sensor
2	bat1-state	-	See section Device Generic : Sensor
3	bat2-voltage	-	See section Device Generic : Sensor
4	bat2-state	-	See section Device Generic : Sensor

9	charger1-voltage	-	See section Device Generic : Sensor
10	charger1-state	-	See section Device Generic : Sensor
13	gps	gps	GPS location of 3G Modem
14	rssi	number	RSSI of 3G Modem
15	ber	number	Bit Error Rate of wireless connection
100	du-ultrasonic	enum (see du-ultrasonic	Ultrasonic object detection. Deprecated use uid 102
101	duu-pos-tilt	struct	See duu-pos-tilt
102	du-counter	number (u32)	Ultrasonic object detection incremental counter

The DUU publishes sensor info after powerup and initialisation:

```
<-- {
        "dinetrpc"    : 1,
        "device:uid"  : "5f5f64695f73696d756c61746f725f5f",
        "pub"         : "sensor:info",
        "time"        : 1444637483000,
        "result"      : [
            {
                "uid"   : 1,
                "label" : "bat1-voltage",
                 "type"  : "number"
             },
             {
                 "uid"   : 2,
                 "label" : "bat1-state",
                 "type"  : "enum",
                 "enum"  : {
                   "empty": 0,
                   "crit" : 1,
                   "low"  : 2,
                   "half" : 3,
                   "full" : 4
                 }
             },
             {
                 "uid"   : 10,
                 "label" : "du-ultrasonic",
                 "type"  : "enum",
                 "enum"  : {
                     "none"  : 0,
                     "left"  : 1,
                     "right" : 2
                     "both"  : 3
                 }
             }
         ]
     }
```

The DUU sensor data publish

```
<-- {
        "dinetrpc"    : 1,
        "device:uid"  : "5f5f64695f73696d756c61746f725f5f",
        "pub"         : "sensor:data",
        "time"        : 1444637483000,
        "result"      : [
            {
                "uid"   : 1,
                "time"  : 1444637481281,
                 "value" : 12.048415652103751
             },
             {
                 "uid"   : 2,
                 "time"  : 1444637480234,
                 "value" : 4
             }
         ]
     }
```

34.3.1. du-ultrasonic

Detection unit ultrasonic state enumerator values:

Enumerator	Value	Description
"err"	-1	Sensor malfunction or read error
"none"	0	No object detection
"single"	1	Single sensor object detection
"both"	2	Both sensors object detection

34.3.2. duu-pos-tilt

DUU position acceleration g-force sensor

Property	Description
"x"	X-axis acceleration in mG
"y"	Y-axis acceleration in mG
"z"	Z-axis acceleration in mG

```
<-- {
          "dinetrpc"    : 1,
          "device:uid"  : "5f5f64695f73696d756c61746f725f5f",
          "pub"         : "sensor:data",
          "time"        : 1444637483000,
          "result"      : [
              {
                  "uid"   : 101,
                  "time"  : 1444637481281,
                   "value" : {
                     "x" : 20,
                     "y" : 20,
                     "z" : 1000,
                   }
              }
          ]
      }
```

34.3.3. duu-pos-rota

DUU position rotation magnetometer sensor gives the amount of degrees the device is pointing to the magnetic north of the earth.

```
<-- {
          "dinetrpc"    : 1,
          "device:uid"  : "5f5f64695f73696d756c61746f725f5f",
          "pub"         : "sensor:data",
          "time"        : 1444637483000,
          "result"      : [
              {
                  "uid"   : 102,
                  "time"  : 1444637481281,
                   "value" : 350.20
               }
          ]
      }
```

34.4. Notify

The available notify info and data for the DUU is listed in the table below:

Unique ID	Label	Type	Description
1-99	-	-	See Chapter Device Generic : Notify
100	du-detection	bool	Detection unit object detected

35. Device Warning Unit Mobile (WUM)

This section describes the capabilities of the Warning Unit Mobile (WUM)

35.1. Device

35.1.1. Version property

Key	Description
`"hw-main"`	WUM main hardware PCB version ('0.0' when unknown)
`"fw-main"`	WUM main MCU firmware version ('0.0.0' when unknown)
`"hw-monitor"`	WUM monitor hardware PCB version ('0.0' when unknown)
`"fw-monitor"`	WUM monitor MCU firmware version ('0.0.0' when unknown)

35.2. Action

The available action info and data for the WUM is listed in the table below:

Unique ID	Label	Type	Description
1-99	-	-	See Section Device Generic : Action
100	wu-state	enum	The WUM warning state signaled from the server (described in wu-state enumeration)

35.2.1. 'wu-state' enumeration

The wu-state enumerator as defined in the table below:

Enumerator	Value
"silent"	0
"detection"	1
"manual"	2
"error"	3

35.3. Sensor

The available sensor info and data for the WUM is listed in the table below:

Unique ID	Label	Type	Description
1	bat1-voltage	number	See section Device Generic : Sensor
2	bat1-state	enum	Battery #1 state (see bat-state enumeration)
3	bat2-voltage	number	See section Device Generic : Sensor
4	bat2-state	enum	Battery #2 state (see bat-state enumeration)
9	charger1-voltage	number	See section Device Generic : Sensor
10	charger1-state	enum	Charger #1 state (see charger-state enumeration)

11	charger2-voltage	number	See section Device Generic : Sensor
12	charger2-state	enum	Charger #2 state (see charger-state enumeration)
13	gps	gps	GPS location of 2G/3G Modem
14	rssi	number	RSSI of 2G/3G Modem
15	ber	number	Bit Error Rate of wireless connection
100	wu-alarm-button	bool	Warning unit manual alarm button

| 101 | wu-alarm-type | enum | Warning unit alarm type (see wu-alarm-type enumeration) | |

35.4. Notify

The available notify info and data for the WUM is listed in the table below:

Unique ID	Label	Type	Description
1-99	-	-	Generic range is unused
100	wu-alarm	bool	The alarm of the warning unit triggered (not manual)
101	wu-alarm-manual	bool	The alarm of the warning unit triggered (manually)

35.5. Config

5 The available configuration info and data is listed in the table below:

Unique ID	Label	Type	Description
1	token	-	See device generic token
2	activate	-	See device generic activate
3	service	-	See device generic service
100	wu-wa-selection	enum	The warning unit warning selection for error and train detection. See wu-wa-selection enumeration
101	wu-volume	number(u8)	The warning unit volume setting
102	wu-volume-reference	number(u16)	reference level for speaker test

35.5.1. wu-wa-selection enumeration

The wu-wa-selection enumerator as defined in the table below:

Enumerator	Value

"wa1"	0
"wa2"	1

35.5.2. wu-alarm-type enumeration

The wu-alarm-type-selection enumerator as defined in the table below:

Enumerator	Value
"no-alarm"	0
"device-error"	1
"server-timeout"	2
"server-error"	3
"server-manual"	4
"server-detection"	5
"manual-button"	6

36. Device ZKL 3000 RC

36.1. Device

36.1.1. Version property

Key	Description
`"hw-main"`	ZKL main hardware PCB version
`"fw-main"`	ZKL main MCU firmware version
`"fw-wcpu"`	ZKL wcpu firmware version (only applicable for ZKLRCv2)
`"hw-switch"`	ZKL switch hardware PCB version (only applicable for ZKLRCv3)
`"fw-switch_control"`	ZKL switch control MCU firmware version (only applicable for ZKLRCv3)
`"fw-switch_meas"`	ZKL switch measurement MCU firmware version
`"fw-switch_drive"`	ZKL switch drive MCU firmware version

36.2. Sensor

The available sensor info and data for the ZKL 3000 RC is listed in the table below:

Unique ID	Label	Type	Description
1	bat1-voltage	number	See section Device Generic : Sensor
2	bat1-state	enum	Battery #1 state (see bat-state enumeration)
3	bat2-voltage	number	See section Device Generic : Sensor
4	bat2-state	enum	Battery #2 state (see bat-state enumeration)
9	charger1-voltage	-	See section Device Generic : Sensor
10	charger1-state	enum	Charger #1 state (see charger-state enumeration)

13	gps	gps	
14	rssi	number	RSSI of 2G/3G Modem
15	ber	number	Bit Error Rate of wireless connection
100	detection-quality	number	Detection quality percentage
101	detection-status	bool	Detection OK?
102	measurement	bool	Measurement on?
103	ba	number	Current B/A value
104	frequency	number	Measurement frequency
105	sw-short	struct (see sections short/battery state)	short state of seperate sections and overall state
106	sw-battery	struct (see sections short/battery state)	battery state of seperate sections and overall state
150	switch-state	bool	Short-circuit enabled?
151	keyswitch	enum (see keyswitch enum)	State of the keyswitch

The ZKL 3000 RC publishes sensor info after connection.

36.2.1. keyswitch enumeration

The keyswitch enumerator as defined in the table below:

Enumerator	Value
"unknown"	0
"operational"	1
"on"	2
"off"	3

36.2.2. sections short/battery state structure

The sections state struct as defined in the table below:

Field	Type
"state"	bool
"section_1"	bool
"section_2"	bool
"section_3"	bool
"section_4"	bool

36.3. Config

The available configuration info and data is listed in the table below:

Unique ID	Label	Type	Description
1	token	-	See Chapter Device Generic : Config
2	activate	-	See Chapter Device Generic : Config
3	service	-	See Chapter Device Generic : Config

100	ba-value	-	Current B/A value
101	frequency	-	Measurement frequency
102	endpoint	string	TCP endpoint location formatted as host:port (e.g `"di-tcp.dualinventive.com:4020"`). Only Legacy devices support this.
103	amplitude	-	Measurement amplitude

37. Device ZKL 3000 RC-C

37.1. Device

37.1.1. Version property

Key	Description
`"hw-main"`	ZKL main hardware PCB version
`"fw-main"`	ZKL main MCU firmware version
`"fw-wcpu"`	ZKL wcpu firmware version (only applicable for ZKLRCv2)
`"hw-switch"`	ZKL switch hardware PCB version (only applicable for ZKLRCv3)
`"fw-switch_meas"`	ZKL switch measurement MCU firmware version
`"fw-switch_drive"`	ZKL switch drive MCU firmware version

37.2. Sensor

The available sensor info and data for the ZKL 3000 RC-C is listed in the table below:

Unique ID	Label	Type	Description
1	bat1-voltage	-	See section Device Generic : Sensor
2	bat1-state	-	See section Device Generic : Sensor
3	bat2-voltage	-	See section Device Generic : Sensor
4	bat2-state	-	See section Device Generic : Sensor
9	charger1-voltage	-	See section Device Generic : Sensor
10	charger1-state	-	See section Device Generic : Sensor
13	gps	gps	GPS location of 2G/3G Modem
14	rssi	number	RSSI of 2G/3G Modem
15	ber	number	Bit Error Rate of wireless connection

100	detection-quality	Number	Detection quality percentage
101	detection-status	Boolean	Detection OK?
102	measurement	Boolean	Measurement on?
150	switch-state	Boolean	Short-circuit enabled?
151	keyswitch	Enum (see keyswitch enum)	State of the keyswitch

The ZKL 3000 RC-C publishes sensor info after connection.

37.2.1. keyswitch enum

The keyswitch enumerator as defined in the table below:

Enumerator	Value
"unknown"	0
"operational"	1
"on"	2
"off"	3

37.3. Config

The available configuration info and data is listed in the table below:

Unique ID	Label	Type	Description
1	token	-	See Chapter Device Generic : Config
2	activate	-	See Chapter Device Generic : Config
3	service	-	See Chapter Device Generic : Config
102	endpoint	string	TCP endpoint location formatted as host:port (e.g `"di-tcp.dualinventive.com:4020"`). Only Legacy devices support this.

38. Device ZKL 3000

38.1. Device

38.1.1. Version property

Key	Description
`"hw-main"`	ZKL main hardware PCB version
`"fw-main"`	ZKL main MCU firmware version
`"fw-wcpu"`	ZKL wcpu firmware version (only applicable for ZKLRCv2)

38.2. Sensor

The available sensor info and data for is listed in the table below:

Unique ID	Label	Type	Description
1	bat1-voltage	number	See section Device Generic : Sensor
2	bat1-state	enum	Battery #1 state (see bat-state enumeration)
3	bat2-voltage	number	See section Device Generic : Sensor
4	bat2-state	enum	Battery #2 state (see bat-state enumeration)
13	gps	gps	GPS location of the device
14	rssi	number	RSSI of 2G/3G Modem
15	ber	number	Bit Error Rate of wireless connection
100	detection-quality	number	Detection quality percentage
101	detection-status	bool	Detection status ok/nok
102	measurement	bool	Measurement status on/off

38.3. Config

The available configuration info and data is listed in the table below:

Unique ID	Label	Type	Description
3	service	-	See device generic service
102	endpoint	string	TCP endpoint location formatted as host:port (e.g "di-tcp.↩ dualinventive.com↩ :4020"). Only Legacy devices support this.

39. Low-level protocol

On the Session and presentation OSI layer the TCP-protocol is described in this section. The TCP packet contains a 6-byte header to distinguish the header of a packet.

39.1. Message header

```
+--------+--------+--------+------+------+------+~~~~~~~~~~~~~~
|  0x44  |  0x4A  |  0x52  | type | packet size | data ...
+--------+--------+--------+------+------+------+~~~~~~~~~~~~~~
```

The first 3 bytes are the DI-Net protocol magic to distinguish the start of a message. Type can be one of the following:

Type	Enumerator	Description
0x01	DNP_HS_REQUEST	Initial handshake
0x02	DNP_REPLY	Response message
0x03	DNP_REGISTER	Register a `device:uid` on current established connection

0x04	DNP_UNREGISTER	Unregister a device:uid on current established connection
0x10	DNP_PLAIN	Unencrypted communication (debug purposes)
0x20	DNP_ENCRYPTED	Encrypted communication
0x40	DNP_TIME	Request DI-Net time (see time)
0xff	-	Reserved for future use

Packet size is a MSB-first unsigned 16-bit number. The length includes the header-size of 6 bytes.

39.2. Initial handshake

The initial handshake exists to announce a device to the server. When a device connects it sends the handshake request. This packet contains nothing more than the device's "uid" as a human readable (hex-encoded) ASCII string (exact 32 bytes). Where only the 0-9 and a-f characters are recommended. Non printable ASCII characters are forbidden (due to JSON/Me1yyssagePack limitation).

39.3. Response message

The reply only contains the ASCII string "WTF" (3 bytes, not null terminated) on failure and replies with random 128-bits of data (16 bytes). After the handshake the device and server are identified to each-other and can continue to communicate. The random data is meant as the challenge for the encryption.

The reply could also contain the ASCII string "MKAY" (4 bytes, not null terminated) when the request is correct.

39.4. Register/Unregister

Register is used for a existing connection to announce a routed device:uid. Unregister is used to announce removal of a routed device:uid for a connection. Both packets contain nothing more than the device's "uid" as a human readable hex-encoded ASCII string (exact 32 bytes). Where only the 0-9 and a-f characters are recommended. Non printable ASCII characters are forbidden (due to JSON/MessagePack limitation).

A device:uid can only be registered after the initial handshake (DNP_HS_REQUEST) was succesfull.

39.5. Unencrypted and Encrypted communication

The communication is encrypted or plain. Plain communication is only used for debugging purposes. On released deployments this packet type is disabled. The communication is CCPSKE-AES-128-CBC-H↩
MAC-MD5-IVFI. Within this packet the MessagePack packet is wrapped.

39.6. Request time

The request time will reply DI-Net time (uint64). The struct is sent in big-endian byte order. See time.

40. Realtime status caching

Realtime status is cached in a Redis NoSQL database. The database layout is mapped 1:1 with the DI-Net RPC protocol.

Databases and schema

The Realtime status cache uses only one database. A Redis cluster only allows a single database. It separates device and project related data with key prefixes.

Update event notifications

For microservices which require realtime update notification and don't poll redis the PUBSUB mechanism of Redis is used. A client is able to subscribe on the following channels:

- The `device` channel where `device:uid` messages are published
- The `project` channel where `project:id` messages are published

Device status

The layout of the device status entries is as follows:

- Hash field `"device:uid"`, e.g: `"device:5f5f64695f73696d756c61746f725f5f"`
  - Key `"last_update"`, last update time when another key in the hash was written
  - Key
    - `"[class]:[method]"` or
    - For devices: `"[class]:[uid]:[method]"` or
  - Value MUST be an JSON object, with added extra key `"last_update"`

Example

The information below is stored in a hash-field

```
last_update      1432734446000
device:info      {"last_update":1432734446000,"label":"tws-3000-wum"}
connection:connect  {"last_update":1432734446000,"time":0.0,"peer":"127.0.0.1:1337"}
connection:disconnect   {"last_update":1432734446000,"time":0.0,"peer":"127.0.0.1:1337","code":0}
dncm:ping        {"last_update":1432734446000,"time":0.0}
sensor:1:info        {"last_update":1432734446000,"uid":1,"label":"bat1-voltage","type":"number"}
sensor:1:data        {"last_update":1432734446000,"uid":1,"time":0.0,"value":6.66}
sensor:2:info        {"last_update":1432734446000,"uid":2,"label":"bat2-voltage","type":"number"}
sensor:2:data        {"last_update":1432734446000,"uid":2,"time":0.0,"value":5.23}
project:3:counter   {"last_update":1432734446000,"id":3,"time":1432734446000,"value":1}
```

Project status

- Hash field `'project:id'`, e.g: `"project:1389"`
  - Key `"last_update"`, last update time when another key in the hash was written
  - Key
    - `"[class]:[method]"` or
    - For projects: `"[class]:[id]:[method]"`
  - Value MUST be an JSON object, with added extra key `"last_update"`

`last_update` key

When writing a key-value to Redis a UNIX timestamp is generated on the server for frontends to know when the server did write the value. The timestamp MUST have a minimal granularity of milliseconds. E.g: `"last_update" : "1432734446000"`.

41. Secure server

The Secure server only communicates projects. Therefore the project class applies here.

The frontend and backend have the same messages, the Secure server forwards certain methods when they are accepted.

The reply of the secure server is a generic empty reply or an error reply. When performing a request that is processed by business-logic the error reply contains a list of errors from the devices, giving the user more diagnostics on the occurred error.

An example of a request message on the frontend is shown below.

```
{
    "dinetrpc":1,
    "id":87,
    "time":1448373137142,
    "project:id":15,
    "user:id" : 3,
    "req":"project:unlock",
    "params":{
        "project":{
            "setup":{
                "project:id":15,
                "type":"tws",
                "roles":{
                    "setup":{
                        "store":[{"user:id":3},{"user:id":20}],
                        "return":[{"user:id":3},{"user:id":20}],
                        "release":[{"user:id":3},{"user:id":20}]
                    },
                    "planning":{
                        "store":[{"user:id":3},{"user:id":20}],
                        "verify":[{"user:id":3},{"user:id":20}],
                        "validate":[{"user:id":3},{"user:id":20}]},
                    "design":{
                        "store":[{"user:id":3},{"user:id":20}],
                        "verify":[{"user:id":3},{"user:id":20}],
                        "validate":[{"user:id":3},{"user:id":20}]
                    }
                }
            },
            "planning":{
                "groups" : [
                    {
                        "id" : 1,
                        "name" : "Kanaal 1",
                        "operators":[
                            {"user:id":3,"from":0,"to":0},
                            {"user:id":20,"from":0,"to":0}
                        ],
                    },
                    {
                        "id" : 2,
                        "name" : "Kanaal 2",
                        "operators":[
                            {"user:id":6,"from":0,"to":0},
                            {"user:id":14,"from":0,"to":0}
                        ],
                    }
                ],
                "devices":[
                    {"device:uid":"001334102700560012513432363637363"},
                    {"device:uid":"001334102700560012513432318437363"}
                ]
            },
            "design":{
                "devices":[
                    {"device:uid":"001334102700340012513432363637363","group:id":1,"role":"strike_in"},
                    {"device:uid":"0013341035004d000951343132353330","group:id":1,"role":"strike_out"}
                    {"device:uid":"001334102700340012513834363637363","group:id":2,"role":"strike_in"},
                    {"device:uid":"0013341035004d000951128332353330","group:id":2,"role":"strike_out"}
                ]
            }
        },
        "devices":[
            {"device:uid":"001334102700560012513432363637363"}
        ]
        "groups" : [
            {"id" : 1},
            {"id" : 2},
        ]
    }
}
```