RED SMART
research for embedded devices

The Electronic Brand of

Integration Manual

Trace&Follow

Energy Monitoring Platform

IDT S.r.l.s.b

Headquarters: Corso Orbassano 402/6 - 10137 - Torino (TO)
Registered office: Via Sebastiano Valfrè 16 - 10121 - Turin (TO)

Tel. +39 011 0922786
E-mail administration: monica.giacomelli@dtsolution.com
E-mail technical department:
alberto.mazzucchelli@idtsolution.com
E-mail business office: margherita.ferragatta@idtsolution.com
Site: https://idtsolution.com/

Rev. 3 - 29-07-2024

Summary

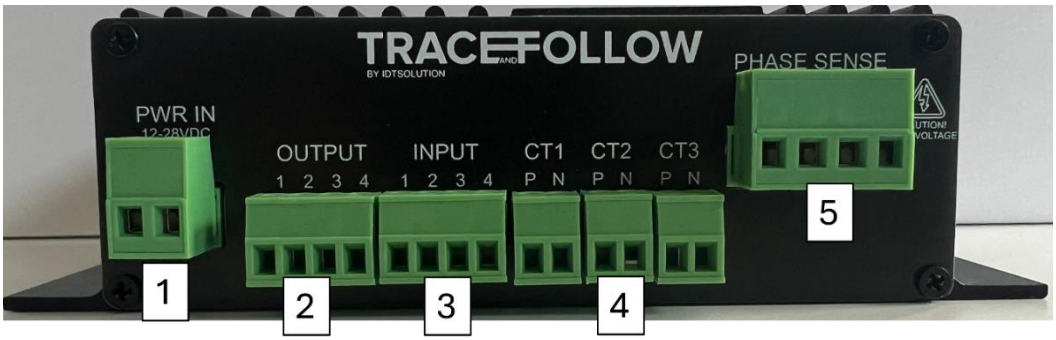
TECHNICAL FEATURES

Dimensions (mm)	150 x 95 x 40
Weight (g)	300
Degree of protection	IP20
Temperature of use	0..+50°
Humidity of use	Maximum 95%, no condensation
Power supply	24 VDC +/- 10% 1 A max.
Communication protocol	Modbus TCP, MQTT, RS485(internal expansions)
Connection	Ethernet, Wi-Fi 2.4g, 4G, LoRaWAN

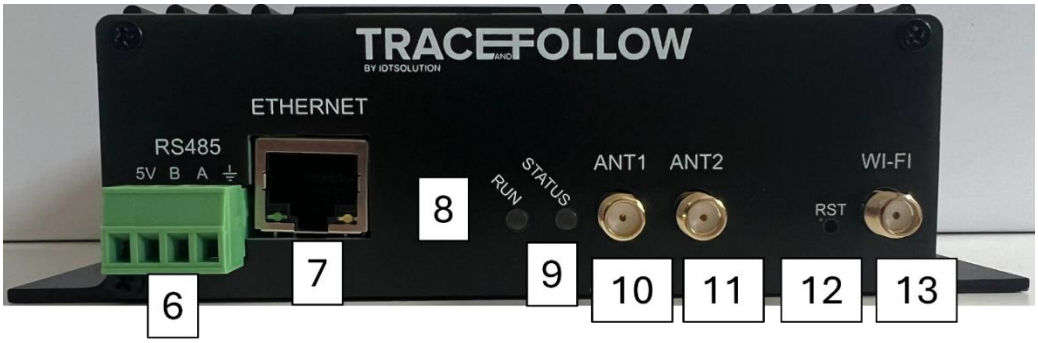
COMPONENT OVERVIEW

HARDWARE COMPONENTS

Boards



1. Power terminal
2. Digital output terminal
3. Digital input terminal
4. Current sensor connection terminals
5. Connection port for "Voltage sampling" expansion



6. RS485 terminal
7. Ethernet connector
8. Led Run
9. Status Led
10. Main antenna connector
11. Auxiliary antenna connector

12. Reset button
13. Wi-Fi antenna connector

14. Ground connection

ASSEMBLY

The device is intended to be installed in either a 72 mm DIN enclosure or a flanged enclosure of the same size as the board. No other installation options are available. Typically, the module is delivered pre-installed in a DIN enclosure or its flanged container.

SUPPLY

The device should be powered at **24 VDC** +/- 10%. The maximum consumption is **1A**, which depends on the type of expansions and cloud connection used. The device is protected against reverse polarity.

FUNCTIONALITY OVERVIEW

Trace&Follow is a device that monitors the energy consumption of industrial machinery and sends the acquired data to a cloud platform. It was designed and developed entirely by RedSmart, a business unit of IDT.

The board's primary function is to measure the power consumption of industrial machinery. It can acquire digital signals and drive digital outputs. The board sends the acquired data to a broker that can be set using the MQTT protocol.

In addition to these basics, the board implements other features:

- **Access-point**: The board can create a Wi-Fi access point and expose a web page to a configurable IP address.
- **Modbus TCP server**: The Trace&Follow board serves as a Modbus server, enabling external devices to read and write data within the board. It operates as a slave within the network and can be accessed by other devices.
- **Expansion modules**: The Trace&Follow board can interface with external modules through an RS485 port.
- **Multiple network interfaces**: The MQTT broker can be accessed through various network interfaces, including Wi-Fi 2.4G, Ethernet, or 4G. Two MQTT interfaces can be configured. In the case of a LoRaWAN connection, the card does not connect directly to an MQTT broker, but sends a radio packet to a LoRa gateway, which decodes it. The connection to the MQTT broker is handled by the gateway.
- **CutOff**: A relay can be connected to one of the digital outputs of the board to interrupt the power supply of the connected machine and turn it off. A power threshold can be set to determine when the machine is in the "idle" state, and a maximum time limit can be set for the machine to remain in this state. If this time limit is exceeded, the Trace&Follow board will switch the relay and shut down the machine.

  Note: If the power supply of the board is the same as that of the machine and no external UPS is provided, the board will also shut down because it does not have a built-in UPS.

The board captures data that is divided into two categories:

- **Periodic data**: Data is sampled at regular intervals, with the time being configurable, and is used to create a data packet that is inserted into the transmission buffer. The data is then published in JSON-formatted messages on the MQTT interface, along with the corresponding timestamp of packet creation.
- **Non-periodic data**: The data is transmitted immediately upon an event trigger, such as a change in input state. Unlike periodic data, it is not accompanied by a timestamp.

SETUP

Connect the power supply, necessary expansions, antennas, and Ethernet cable if required.

Configuration files

Device configuration is accomplished by uploading a configuration file directly to the Trace&Follow board through its web interface.

The file must be named **"TF_config.json"** and be in JSON format, containing the required fields:

- **"T&F config file:**
  - Description: object containing configuration parameters
  - Type: JSON object
  - Fields:
    1. **"CREDENTIALS:**
       - Description: object containing configuration parameters for accessing functions that require credentials in the configuration web page
       - Type: JSON object
       - Fields:
         (a) **"user":**
           - Description: username
           - Type: String
         (b) **"password"**
           - Description: password
           - Type: String
    2. **"AP."**
       - Description: object containing the configuration parameters of the Wi-Fi access-point generated by the board.
       - Type: JSON object
       - Fields:
         (a) **"name":**
           - Description: name of the Wi-Fi network generated by the board
           - Type: String
         (b) **"password"**
           - Description: Wi-Fi network password (8 to 63 characters)
           - Type: String
         (c) **"ip":**
           - Description: IP address of the web page of the board
           - Type: String
         (d) **"mask"**
           - Description: network mask
           - Type: String
    3. **"WIFI:**
       - Description: object containing the configuration parameters of the Wi-Fi connection
       - Type: JSON object
       - Fields:
         (a) **"use_dhcp":**
           - Description: DHCP enabled
           - Type: Bool
         (b) **"ip":**
           - Description: fixed IP address of the board (used if use_dhcp = 0)
           - Type: String (ex: "192.168.1.100")

(c) **"mask"**
- Description: network mask (used if use_dhcp = 0)
- Type: String

(d) **"gateway"**
- Description: network gateway (used if use_dhcp = 0)
- Type: String

(e) **"dns":**
- Description: network dns (used if use_dhcp = 0)
- Type: String

(f) **"ssid":**
- Description: name of the network to connect to
- Type: String

(g) **"password"**
- Description: password of the network to connect to
- Type: String

(h) **"channel"**
- Description: Wi-Fi network channel
- Type: Integer [0..14, 0=auto]

(i) **"bssid":**
- Description: bssid of the Wi-Fi network
- Type: String

4. **"RTC":**
- Description: object containing the configuration parameters of the RTC
- Type: JSON object
- Fields:

(a) **"time_from_network":**
- Description: timestamp initialization configuration
- Type: Bool

(b) **"timezone"**
- Description: timezone to be used for the board
- Type: Integer [-12..+12]

(c) **"timestamp_rel":**
- Description: forces the publication of messages with relative timestamps related to time at which the board is turned on instead of absolute timestamps
- Type: Bool

5. **"ETH"**
- Description: object containing the configuration parameters of the Ethernet connection
- Type: JSON object
- Fields:

(a) **"use_dhcp":**
- Description: DHCP enabled
- Type: Bool

(b) **"ip":**
- Description: fixed IP address of the board (used if use_dhcp = 0)
- Type: String (ex: "192.168.1.100")

(c) **"mask"**
- Description: network mask (used if use_dhcp = 0)
- Type: String

(d) **"gateway"**
- Description: network gateway (used if use_dhcp = 0)
- Type: String

(e) **"dns":**
- Description: network dns (used if use_dhcp = 0)
- Type: String

6. **"MAIN_MQTT"**
   - Description: object containing the configuration parameters of the connection to the MQTT broker (used in case of Wi-Fi, Ethernet and 4G connection)
   - Type: JSON object
   - Fields:
     - **(a) "server"**
       - Description: MQTT server to connect to.
       - Type: String
     - **(b) "port"**
       - Description: server port to connect to
       - Type: integer [1..65535]
     - **(c) "attribute":**
       - Description: MQTT topic on which to publish attributes.
       - Type: String (ex: "topic/attribute")
     - **(d) "telemetry"**
       - Description: MQTT topic on which to publish telemetry data.
       - Type: String
     - **(e) "rpc":**
       - Description: MQTT topic to post to from which the board receives rpc commands
       - Type: String
     - **(f) "username":**
       - Description: username MQTT server authentication
       - Type: String
     - **(g) "password"**
       - Description: MQTT server authentication password
       - Type: String
     - **(h) "connection_int":**
       - Description: time interval (msec) between reconnection attempts in case of connection error
       - Type: Integer [0..65535]

7. **"SECONDARY_MQTT"**
   - Description: object containing the configuration parameters of the connection to the secondary MQTT broker (used in case of Wi-Fi, Ethernet and 4G connection)
   - Type: JSON object
   - Fields:
     - **(a) “active”:**
       - Description: enable secondary MQTT.
       - Type: Bool
     - **(b) "server"**
       - Description: MQTT server to connect to.
       - Type: String
     - **(c) "port"**
       - Description: server port to connect to
       - Type: integer [1..65535]
     - **(d) "attribute":**
       - Description: MQTT topic on which to publish attributes.
       - Type: String (ex: "topic/attribute")
     - **(e) "telemetry"**
       - Description: MQTT topic on which to publish telemetry data.
       - Type: String
     - **(f) "rpc":**
       - Description: MQTT topic to post to from which the board receives rpc commands
       - Type: String
     - **(g) "username":**
       - Description: username MQTT server authentication
       - Type: String

(h) "password"
- Description: MQTT server authentication password
- Type: String

(i) "connection_int":
- Description: time interval (msec) between reconnection attempts in case of connection error
- Type: Integer [0..65535]

8. "LORA":
- Description: object containing LoRa connection configuration parameters. Leave fixed as in the example
- Type: JSON object
- Fields:

(a) "app_eui":
- Description: eui LoRa app
- Type: String

(b) "app_key":
- Description: app key LoRa
- Type: String

(c) }, "downlink_time":
- Description: reception waiting time after a sending
- Type: Integer [0..10000]

(d) "max_send_size":
- Description: max number of bytes per message. To be set fixed at 65
- Type: integer [65]

(e) "power"
- Description: transmission power index.
- Type: Integer [0..5]

(f) "min_transmission_time":
- Description: minimum time between two consecutive transmissions in milliseconds. Min:15000
- Type: Integer [15000..60000]

9. "HTTP" (not used):
- Description: object containing HTTP connection configuration parameters
- Type: JSON object
- Fields:

(a) No fields are present, reserved for future implementation

10. "SD" (not used):
- Description: object containing configuration parameters of SD card usage
- Type: JSON object
- Fields:

(a) No fields are present, reserved for future implementation

11. "CONNECTION:
- Description: object containing connection type configuration parameters
- Type: JSON object
- Fields:

(a) "type":
- Description: connection type (1-> Ethernet, 2-> WiFi, 3-> LoRa, 4-> 4G)
- Type: integer

(b) "machine" (not used):
- Description: reserved (to be left fixed at 1)
- Type: integer

12. **"PACKET":**
    - Description: object containing package creation configuration parameters
    - Type: JSON object
    - Fields:
        - **(a) "packet_int":**
            - Description: packaging time in milliseconds
            - Type: integer [1000..4294967295]
        - **(b) "buffer_size":**
            - Description: maximum number of packets that can be saved in the transmission buffer
            - Type: integer

13. **"POWER."**
    - Description: object containing energy measurement configuration parameters
    - Type: JSON object
    - Fields:
        - **(a) "sampling_int":**
            - Description: time between energy meter sampling in milliseconds
            - Type: integer
        - **(b) "voltage_sensor":**
            - Description: use of voltage sampling. 0-> no, 1-> yes
            - Type: Bool
        - **(c) "line_freq":**
            - Description: grid frequency (value used if voltage_sensor = 0, otherwise detected by the board)
            - Type: integer
        - **(d) "pga_gain":**
            - Description: Value to be set according to the CT used. See the relevant table[1].
            - Type: integer
        - **(e) "voltage_gain":**
            - Description: input gain of PGA on voltages
            - Type: integer
        - **(f) "gain_ct1":**
            - Description: CT1 calibration. Value to be set according to the CT used. See the relevant table[1].
            - Type: integer
        - **(g) "gain_ct2":**
            - Description: CT2 calibration. Value to be set according to the CT used. See the relevant table[1].
            - Type: integer
        - **(h) "gain_ct3":**
            - Description: CT3 calibration. Value to be set according to the CT used. See the relevant table[1].
            - Type: integer
        - **(i) "pf":**
            - Description: power factor (value used if voltage_sensor = 0, otherwise detected by the board)
            - Type: Float
        - **(j) "voltage"**
            - Description: line voltage (value used if voltage_sensor = 0, otherwise detected by the board)
            - Type: integer
        - **(k) "connection_type":**
            - Description: type of connection: 0 -> three-phase with Neutral, 1 -> three-phase without Neutral
            - Type: integer

14. **"INPUTS:**
    - Description: object containing configuration parameters of physical and virtual inputs
    - Type: JSON object
    - Fields:
        - **(a) "number_phy":**
            - Description: number of physical inputs. To be set to 4
            - Type: integer
        - **(b) "inputs"**
            - Description: array of physical input description. Must contain number_phy elements
            - Type: Array JSON
            - Fields:
                - **a. "num":**
                    - Description: input number, must contain a sequence number for each array element starting from number 1
                    - Type: integer
                - **b. "enabled"**
                    - Description: enabling input
                    - Type: Bool
                - **c. "mod":**
                    - Description: input mode. (1-> normal, 2-> counter, 3-> function, 4-> blinking)
                    - Type: integer
                - **d. "polarity"**
                    - Description: input polarity (0-> normal, 1-> inverse)
                    - Type: Bool
                - **e. " filter_time":**
                    - Description: input filter, no pulse below this threshold is considered in mode 1
                    - Type: integer [milliseconds]
                - **f. “reset_counter”:**
                    - Description: Enabling reset of the counter associated with the input after sending when it is configured in "counter" mode(2)
                    - Type: Bool
                - **g. “add_to_packet”:**
                    - Description: Enables the insertion of the input value, if configured in mode 1, 3 or 4, within the transmitted periodic data packet. When enabled, asynchronous messages related to status changes of this input are no longer sent.
                    - Type: Bool

        - **(c) "number_virt":**
            - Description: number of virtual inputs. To be set according to how many virtual inputs you want to set.
            - Type: integer
        - **(d) "virtual_inputs":**
            - Description: virtual input description array. Must contain **number_virt** elements
            - Type: Array JSON
            - Fields:
                - **a. "num":**
                    - Description: input number, must contain a sequence number for each array element starting from number 1
                    - Type: integer
                - **b. "enabled":**
                    - Description: enabling input
                    - Type: Bool
                - **c. "mod":**
                    - Description: input mode. (1-> normal, 2-> counter, 3-> function, 4-> blinking)
                    - Type: integer

d. **"polarity":**
   - Description: input polarity (0-> normal, 1-> inverse)
   - Type: Bool
e. **"filter_time":**
   - Description: input filter, no pulse below this threshold is considered in mode 1
   - Type: integer [milliseconds]
f. **“reset_counter”:**
   - Description: Enabling reset of the counter associated with the input after sending when it is configured in "counter" mode(2)
   - Type: Bool
g. **“add_to_packet”:**
   - Description: Enables the insertion of the input value, if configured in mode 1, 3 or 4, within the transmitted periodic data packet. When enabled, asynchronous messages related to status changes of this input are no longer sent.
   - Type: Bool

15. **"OUTPUTS:**
- Description: object containing configuration parameters of physical and virtual outputs
- Type: JSON object
- Fields:

(a) "number_phy":
- Description: number of physical inputs. To be set to 4
- Type: integer

(b) "outputs"
- Description: array of description of physical outputs. Must contain number_phy elements
- Type: Array JSON
- Fields:

a. **"num":**
   - Description: input number, must contain a sequence number for each array element starting from number 1
   - Type: integer
b. **"polarity"**
   - Description: input polarity (0-> normal, 1-> inverse)
   - Type: Bool
c. **"pin"**
   - Description: pin input. (fixed: input1=47, input2 = 46, input1=45, input2 = 44)
   - Type: integer

(c) "number_virt":
- Description: number of virtual outputs. To be set according to how many virtual outputs you want to set.
- Type: integer

(d) "virtual_inputs":
- Description: virtual output description array. Must contain number_virt elements
- Type: Array JSON
- Fields:

a. **"num":**
   - Description: output number, must contain a sequence number for each array element starting from number 1
   - Type: integer
b. **"polarity":**
   - Description: input polarity (0-> normal, 1-> inverse)
   - Type: Bool

c. **"pin":**
   - Description: confidential, leave fixed at 0
   - Type: integer

16. **"CUTOFF:**
    - Description: object containing the configuration parameters of the CutOff functionality
    - Type: JSON object
    - Fields:
      (a) **"is_active":**
        - Description: enable CutOff
        - Type: Bool
      (b) **"pin":**
        - Description: pin relay cutoff (fixed: input1=47, input2 = 46, input1=45, input2 = 44)
        - Type: Integer
      (c) **"threshold:**
        - Description: threshold power
        - Type: Integer [W]
      (d) **"idle_time":**
        - Description: time after which relay is triggered if idle
        - Type: Integer [milliseconds]

17. **"LOCALIZATION."**
    - Description: object containing the configuration parameters of the Localization feature
    - Type: JSON object
    - Fields:
      (a) **"is_active":**
        - Description: enabling localization
        - Type: Bool

18. **"MODBUS_TCP":**
    - Description: object containing the configuration parameters of the Localization feature
    - Type: JSON object
    - Fields:
      (a) **"is_active":**
        - Description: enable modbus TCP server
        - Type: Bool
      (b) **"input_registers_number":**
        - Description: number input registers.
        - Type: Integer
        - Allowed values: the value should be left fixed at 50
      (c) **"holding_registers_number":**
        - Description: number holding registers
        - Type: Integer
        - Allowable values: the value should be left fixed at 60
      (d) **"coils_number":**
        - Description: number coils
        - Type: Bool
        - Allowed values: the value should be left fixed at 0
      (e) **"is_active":**
        - Description: enabling localization
        - Type: Bool
      (f) **"port":**
        - Description: port on which the server is active
        - Type: Integer

(g) "slave_id":
- Description: modbus id
- Type: Integer

(h) "send_input":
- Description: if = 1, input registers are also sent to the cloud platform
- Type: Bool

(i) "max_client_number":
- Description: maximum number of clients that can be simultaneously connected to the server
- Type: Integer

(j) "idle_timeout":
- Description: time after which, if no requests are made from the client to the server, the client is disconnected
- Type: Integer [millisecond]

Below is the default configuration file:

```
{
  "T&F config file": {
    "CREDENTIALS": {
      "}, "user",
      "password": "password"
    },
    "AP": {
      "name": "TraceAndFollow",
      "password": "password-TF",
      "ip": "192.168.1.1",
      "mask": "255.255.255.0",
      "gateway": "192.168.1.1"
    },
    "WIFI": {
      "use_dhcp": false,
      "ip": "192.168.1.100",
      "mask": "255.255.255.0",
      "gateway": "192.168.1.1",
      "dns": "192.168.1.1",
      "ssid": "ssid",
      "password": "password",
      "channel": 0,
      "bssid": "0"
    },
    "RTC": {
      "time_from_network": false,
      "timezone": 0,
      "timestamp_rel": true
    },
    "ETH": {
      "use_dhcp": false,
      "ip": "192.168.1.100",
      "mask": "255.255.255.0",
      "gateway": "192.168.1.1",
      "dns": "192.168.1.1"
    },
    "MAIN_MQTT": {
      "server": "broker.hivemq.com",
      "port": 1883,
      }, "attribute": "attributes",
```

```
  "telemetry": "telemetry",
  "}, "rpc",
  "username": "user",
  "password": "password",
  "connection_int": 2000
},
"SECONDARY_MQTT": {
      "server": "broker.hivemq.com",
      "port": 1883,
      "attribute": "attributes",
      "telemetry": "telemetry",
      "rpc": "rpc",
      "username": "user2",
      "password": "password2",
      "connection_int": 2000
},
"LORA": {
  "app_eui": "00000000000000000000000000000000",
  "app_key": "00000000000000000000000000000000",
  "downlink_time": 2000,
  "max_send_size": 65,
  "power": 5,
  "min_transmission_time": 30000
},
"HTTP": {},
"SD": {},
"CONNECTION": {
  "type": 1,
  "machine":3
},
"PACKET": {
  "packet_int": 30000,
  "buffer_size": 100
},
"POWER": {
  "sampling_int": 100,
  "voltage_sensor": 0,
  "line_freq": 50,
  "pga_gain": 42,
  "voltage_gain": 1,
  "gain_ct1": 38904,
  "gain_ct2": 38904,
  "gain_ct3": 38904,
  "pf": 0.95,
  "voltage": 230,
  "connection_type": 0
},
"INPUTS": {
  "number_phy": 4,
  "inputs": [
    {
      "num": 1,
      "enabled": false,
      "mod": 1,
      "polarity": 0,
      "filter_time": 1000,
      "reset_counter": false,
      "add_to_packet": false
```

```
      },
      {
        "num": 2,
        "enabled": false,
        "mod": 1,
        "polarity": 0,
        "filter_time": 5000,
        "reset_counter": false,
        "add_to_packet": false
      },
      {
        "num": 3,
        "enabled": false,
        "mod": 1,
        "polarity": 0,
        "filter_time": 1000,
        "reset_counter": false,
        "add_to_packet": false
      },
      {
        "num": 4,
        "enabled": false,
        "mod": 1,
        "polarity": 0,
        "filter_time": 1000,
        "reset_counter": false,
        "add_to_packet": false
      }
    ],
    "number_virt": 1,
    "virtual inputs": [
      {
        "num": 1,
        "enabled": false,
        "mod": 1,
        "polarity": 0,
        "filter_time": 1000,
        "reset_counter": false,
        "add_to_packet": false
      }
    ]
  },
  "OUTPUTS": {
    "number_phy": 4,
    "outputs": [
      {
        "num": 1,
        "polarity": 0,
        "pin": 47
      },
      {
        "num": 2,
        "polarity": 0,
        "pin": 46
      },
      {
        "num": 3,
        "polarity": 0,
```

```
        "pin": 45
      },
      {
        "num": 4,
        "polarity": 0,
        "pin": 44
      }
    ],
    "number_virt": 1,
    "virtual outputs": [
      {
        "num": 1,
        "polarity": 0,
        "pin": 0
      }
    ]
  },
  "CUTOFF": {
    "is_active": false,
    "pin": 44,
    "threshold": 200,
    "idle_time": 100000
  },
  "LOCALIZATION": {
    "is_active": false
  },
  "MODBUS_TCP": {
    "is_active": false,
    "input_registers_number": 50,
    "holding_registers_number": 60,
    "coils_number": 0,
    "port": 502,
    "slave_id": 1,
    "send_input": false,
    "max_client_number": 2,
    "idle_timeout": 600000
  }
 }
}
```


1. **Current Transformers calibration values and pga.**

Model	Report	Voltage/Current	gain_ctx	pga_gain	Voltage_gain
SCT024TS	200A/100mA	C	31595	21	46604
SCT013	100A/1V	T	44409	21	46604

OPERATIONAL STEPS.

Preliminary Checks

Upon turning on the board, verify that the LEDs depicted in the figure are illuminated as indicated. If they are not, inspect the power supply to the board.

Initialization

Upon startup, the Trace&Follow board reads the saved configuration file and configures all devices accordingly. It then initiates the Wi-Fi interface to generate its network and activates the selected network interface for connecting to the cloud platform.
After these steps are completed, the board's "run" LED will flash at a frequency of 1Hz.

Simultaneously, the board's Status LED will glow white, indicating that it is attempting to connect to the cloud via the selected interface.

During the initialization phase, check that the following LEDs are on:

Led **RUN**: flashing at a frequency of 1HZ Led **STATUS**: fixed color WHITE

Ethernet

Possible errors

- STATUS led in red color: Ethernet cable not properly plugged in or link missing.

Wi-Fi

Possible errors

- STATUS led in red color: The device is currently unable to connect to the selected network. After a brief period, the LED will return to white, indicating that a new connection attempt is being made.

4G

Possible errors

- STATUS led in red color: The device is currently unable to connect to the cellular network. After a brief period, the LED will return to white, indicating that a new connection attempt is being made.
  In this case check:
  - That the antenna is properly connected.
  - That the SIM card is inserted of the appropriate slot.
  - That the SIM has an active data plan.

Errors common to the three interfaces:

- RUN led steady on or off: Generic board error. Check the power supply and perform a board reset. If the problem persists, contact technical support.
- STATUS led off: Generic board error. Check the power supply and perform a board reset. If the problem persists contact technical support.

After establishing the connection, the LED turns blue to indicate that the board is connected to the internet.
The board will then attempt to retrieve the current timestamp from the network if needed (refer to chapter “Periodic Data Payload”). If an error occurs, the STATUS LED will flash blue and red alternately. If the timestamp cannot be obtained after three failed attempts, the board will automatically restart the network interface.

The board attempts to connect to the configured MQTT broker.
If there is an error, the STATUS LED flashes red. The reconnection interval can be set in the configuration file. After successfully connecting to the MQTT broker, the STATUS LED turns green, and the board begins sending data.

Note: Obtaining the timestamp and connecting to the broker can be done quickly, causing the STATUS LED to change from white to green.

MQTT INTERFACE

The board can send and receive data to and from a platform using an MQTT connection. The payloads exchanged are in JSON format.

The data collected by the board are divided into two separate categories:

- Non-periodic configuration data, posted in the topic **"attribute"**
- Periodic and non-periodic state data, posted in the topic **"telemetry"**

The data sent to the board should be posted in the **"rpc"** topic, following a format explained below.

All three of these topics are configurable through the configuration file.
The board supports connection to two MQTT brokers simultaneously, each of which can be configured separately from the configuration file. The first one (MAIN_MQTT) is enabled by default and cannot be disabled, the second one (SECONDARY_MQTT) must be enabled via configuration file. Both connections are parallel and independent. OTA updates are only possible through the primary broker.

Periodic data are captured and saved at regular intervals in data structures called “packets”. The board as soon as it has a packet ready attempt to send it to the MQTT broker. In case of sending failure, this packet is saved inside an internal buffe. When communication is restored, it sends all the packets in the buffer. If the resulting payload size is too large, the sending is split into multiple messages until the buffer is empty.
Non-periodic data are sent to the broker as soon as a change in their status is detected.

Data sent from the board

Non-periodic data:

- **input(N)_type**: field indicating the mode in which a physical input is configured.
  - Type: Integer
  - Allowed values: [1-4]

- **virtual_input(N)_type**: field indicating the mode in which a virtual input is configured.
  - Type: Integer
  - Allowed values: [1-4]

- **packet_time**: field indicating how often a packet containing periodic data is created in seconds.
  - Type: Integer
  - Allowed values: [1 ...] seconds
- **energy_standby_threshold**: field indicating the instantaneous power threshold below which the board considers the machine to be in idle state.
  - Type: Integer
  - Allowable values: [0 ...] [W]

- **inactivity_time**: field indicating after how long the CutOff relay is triggered when the machine is in idle state in seconds. This function must be enabled from the configuration file.
    - Type: Integer
    - Allowed values: [15 ...] [s], if = 0, disabled

- **transmission_time**: fixed value of 0.
    - Type: Integer
    - Allowed values: 0

- **hardware_version**: field indicating the hardware version of the board.
    - Type: String

- **firmware_version**: field indicating the firmware version of the board.
    - Type: String

- **connection_type**: field indicating the network interface through which the board is connected to the broker.
    - Type: Integer
    - Allowable values: [1-4]

- **timezone:** field indicating the timezone in which the board is located. The board does not take it automatically but is set by configuration file.
    - Type: Integer
    - Allowable values [-12 - +12]

- **timestamp_rel:** field indicating whether the board is using a relative or absolute timestamp for posting messages.
    - Type: Bool

- **ip**: field indicating the IP of the board in the network.
    - Type: String

- **gateway**: field indicating the gateway configured in the network.
    - Type: String

- **mask**: field indicating the configured network mask.
    - Type: String

- **dns**: field indicating the configured dns.
    - Type: String

- **mac**: field indicating the mac address of the network board.
    - Type: String

Non-periodic state data:

- **input(N)_value**: field indicating the current state of the input or its associated function when input(N)_type = 1 (normal), 3 (function), 4 (blinking)
    - physical input state.
    - Type: Bool
    - Allowable values: [0-1]

- **virtual_input(N)_value**: field indicating the current state of the input or its associated function when virtual_input(N)_type = 1 (normal), 3 (function), 4 (blinking)
    - virtual input state.

  - Type: Bool
  - Allowable values: [0-1]

- **output(N):** physical output state.
  - Type: Bool
  - Allowable values: [0-1]

- **virtual_output(N):** virtual output state
  - Type: Bool
  - Allowable values: [0-1]

Periodic data:

- **energy_consuption**: active energy consumed over the time span of a packet.
  - Type: Float [Wh]

- **input(N)_value**: field indicating the number of pulses received by the related input when input(N)_type = 2 (counter)
  - Number of pulses in the time span of a packet.
  - Type: Integer
  - Allowed values: [0 ...]

- **Virtual_input(N)_value**: field indicating the number of pulses received by the related input when virtual_input(N)_type = 2 (counter)
  - Number of pulses in the time span of a packet.
  - Type: Integer
  - Allowed values: [0 ...]
- **voltage1**: Phase 1 voltage.
  - Type: Float [V]
- **voltage2**: Phase 2 voltage.
  - Type: Float [V]
- **voltage3**: Phase 3 voltage.
  - Type: Float [V]
- **current1**: Phase 1 current.
  - Type: Float [A]
- **current2**: Phase 2 current.
  - Type: Float [A]
- **current3**: Phase 3 current.
  - Type: Float [A]
- **current_tot**: total current across all phases.
  - Type: Float [A]
- **active_power1**: phase 1 active power.
  - Type: Float [W]
- **active_power2**: phase 2 active power.
  - Type: Float [W]
- **active_power3**: phase 3 active power.
  - Type: Float [W]
- **active_power_tot**: total instantaneous active power across all phases.
  - Type: Float [W]
- **reactive_power1**: phase 1 reactive power.
  - Type: Float [VAR]
- **reactive_power2**: phase 2 reactive power.
  - Type: Float [VAR]

- **reactive_power3**: phase 3 reactive power.
  - Type: Float [VAR]
- **reactive_power_tot**: total instantaneous reactive power across all phases.
  - Type: Float [VAR]
- **pf1**: Phase 1 instantaneous power factor.
  - Type: Float
- **pf2**: Phase 2 instantaneous power factor.
  - Type: Float
- **pf3**: Phase 3 instantaneous power factor.
  - Type: Float
- **pf_tot**: Total instantaneous power factor across all phases.
  - Type: Float
- **phase1**: Phase 1 phase shift.
  - Type: Float [°]
- **phase2**: Phase 2 phase shift.
  - Type: Float [°]
- **phase3**: Phase 3 phase shift.
  - Type: Float [°]
- **Frequency:** Grid frequency.
  - Type: Float [Hz]
- **reactive_energy_consuption**: Reactive energy consumed over the time span of a packet.
  - Type: Float [VARh]

Note: When an input is set to mode 1, 3, or 4, it will be sent as soon as a change in state is detected, following the logic of nonperiodic data (if key “add_to_packet" = “false”).

Note: Qualitative energy data is only invoked if the power phases are connected to the board and in the configuration file the key "voltage_sensor" has value "true".

Payload on first connection

Once the board starts and establishes a connection to the broker, it immediately publishes a message on the topic **“attribute”** with the following fields:

- packet_time
- transmission_time
- energy_standby_threshold
- inactivity_time
- hardware_version
- firmware_version
- connection_type
- timezone
- timestamp_rel
- input1_type
- input2_type
- input3_type
- input4_type
- virtual_input(N)_type (one key for each configured virtual input)

Example of initial payload, with topic **attribute**→ v1/devices/me/attributes :

Topic: v1/devices/me/attributes QoS: 0


```
{
  "packet_time": 30,
  "transmission_time": 0,
  "energy_standby_threshold": 5000,
  "inactivity_time": 100,
  "hardware_version": "2.1",
  "firmware_version": "1.0.3",
  "connection_type": 1,
  "timezone": 1,
  "timestamp_rel": false,
  "input1_type": 1,
  "input2_type": 0,
  "input3_type": 0,
  "input4_type": 0
}
```


Network data payload

Whenever the board connects to the MQTT broker, whether it"s the first connection or after a disconnection, it sends a payload with the following fields to the topic **"attribute"**:

- ip
- gateway
- mask
- dns
- mac

Topic: v1/devices/me/attributes QoS: 0


```
{
  "ip": "192.168.1.24",
  "gateway": "192.168.1.1",
  "mask": "255.255.255.0",
  "dns": "192.168.1.1",
  "mac": "B0:B2:1C:D4:50:B7"
}
```


Periodic data payload

Once a packet is prepared and the connection with the broker is established, the board publishes a payload on the **"telemetry"** topic. This payload contains all periodic data for the packet in the form of a JSON array, with each packet represented by a JSON object.
Each object contains two keys:

- **"ts":** timestamp associated with the creation of the package in UNIX format in milliseconds
  - Type: String
- **"values":** object containing all the keys of the package
  - Type: JSON object

The Trace&Follow board has a battery-backed RTC that allows it to maintain the current time even in the event of a power failure.

Depending on the configuration file settings, the board can use the timestamp in the RTC, obtain the current timestamp from the network, or use one related to the startup of the board:

- "timestamp_rel" = true: the board will publish messages using the timestamp related to startup.
- "timestamp_rel" = false and "time_from_network" = true: the board will get the timestamp from the network and will automatically update the one in the RTC and use it for messages.
- "timestamp_rel" = false and "time_from_network" = false: the board the board will use the timestamp saved in the RTC and use it for messages.

Note: f RTC data is lost, such as due to battery removal or discharge, the board will use a relative timestamp. The initial instant will be the startup of the board.

Example of message containing a single package, with topic **telemetry**→ v1/devices/me/telemetry:

Topic: v1/devices/me/telemetry QoS: 0


```
[
  {
    "ts": "1709196304146",
    "values": {
      "energy_consumption": 0.497999996,
      "input2_value": 12,
      "virtual_input1_value": 0
    }
  }
]
```


Example of a message with multiple packets:

Topic: v1/devices/me/telemetry QoS: 0


```
[
  {
    "ts": "1709196514146",
    "values": {
      "energy_consumption": 0.497999996,
      "input2_value": 31,
      "virtual_input1_value": 0
    }
  },
  {
    "ts": "1709196544146",
    "values": {
      "energy_consumption": 0.497999996,
      "input2_value": 0,
      "virtual_input1_value": 0
    }
  },
  {
    "ts": "1709196393401",
    "values": {
      "energy_consumption": 0.499000013,
      "input2_value": 16,
      "virtual_input1_value": 0
    }
  }
]
```

Payload non-periodic state data

If the input is configured in mode 1, 3, or 4, the status change is sent to the **"telemetry"** topic without the associated timestamp, always within a JSON array.

Example of input configured in mode 1, with topic **telemetry**→ v1/devices/me/telemetry:

Non-periodic configuration data payload

In case of non-periodic data changed during the operation of the board, it will be sent on the topic **"attribute"** within a JSON object.

Example with topic **attribute**→ v1/devices/me/attributes:

Data sent from the server

The server responsible for handling the data sent by the board must also send data.
Server-side data submission can occur in two ways:

- **Server first**: in this case the server will send some configuration parameters that the board is ready to receive.
- **Board first**: in this case the board perform a request to the server, which must respond with the requested data.

Server first

The server determines when to send data to the board using the "rpc" topic with the suffix "/request/$request_id".
Example with topic **rpc**→ v1/devices/me/rpc/

v1/devices/me/rpc/request/$request_id

Where $request_id is an integer that identifies the single request made. This id must increment with each request.
When the board makes a new connection the request_id must restart at 0.

The payload sent by the server must consist of a JSON object containing:

- The **"method"** key:
  - Type: String
  - Description: the method to be performed by the board
- The key **"params":**
  - Type: JSON object
  - Description: contains keys with parameters for rpc execution.

Example of an Rpc request made by the server:

method	params	Description
setInputFunctionState	**number:** input number state**:** new state of the function associated with the input	Sets the state of the function associated with the selected input when configured in input mode
setConfigOutput	**number:** output number **polarity:** polarity of the output [0 normal 1 inverted].	Sets the polarity of the selected output
setOutput	number: output number **state:** new output state **time**: time the output remains active (0 = infinity) [50 ... infinity] milliseconds	Sets the status of the selected output. The “time” parameter can be used to send a pulse of a specific duration
setThreshold	**value**: new threshold value for standby [> 0] W	Sets the instantaneous power threshold used in the CutOff function
setInactivityTime	**value**: new idle time (0 = disabled) [30 ... infinity] seconds	Set the maximum idle time in the CutOff function
setPacketTime	**value**: new packet time [1 ...] s	Sets the time between the creation of one package and the next one
reset	**value**: 1	Performs a board reset
setInputRegister	**number:** register number [0-9] **value:** value to be entered into the register [0-65535]	Sets the value of an input register associated with the modbus TCP server managed by the board
setCurrentTimestamp	**Value:** GMT timestamp in UTC format, in seconds	Updates the timestamp value saved within the board RTC

Once the board successfully receives the RPC request, it publishes the same payload received from the server on the "rpc" topic with the suffix /response/$request_id.

Example with topic **rpc**→ v1/devices/me/rpc/

"rpc"/response/$request_id

Where the $request_id parameter has the same value as the one received.

Sample response sent from the board:

Topic: v1/devices/me/rpc/response/1 QoS: 0

```
{
  "method": "setPacketTime",
  "params": {
    "value": 200
  }
}
```

When a parameter is modified, the board will post a message in the **"attribute"** topic that includes the updated value.

Note: In case you go to change the packet time (via RPC), the board will interrupt the current packet, send it to the server, and start a new one with the new time set.

Board first

In this mode, the board requests keys from the server, which must respond with a message in JSON format containing the values of the requested keys.
The request is published on the **"attribute"** topic with the suffix **"/request/$request_id"**.

Example with topic **attribute**→ v1/devices/me/attributes/

v1/devices/me/attributes/request/$request_id

Where the $request_id parameter is an incremental integer that identifies the request.

The request message from the board consists of a JSON object containing the following key:

- **"sharedKeys:**
  - Type: String
  - Description: contains the keys of the required parameters separated by a comma

Sample application:

Topic: v1/devices/me/attributes/request/0 QoS: 0

```
{
  "sharedKeys": "fw_title,fw_version,s_packet_time,s_energy_standby_threshold,s_inactivity_time"
}
```

Server response must be posted on the topic **"attribute"** with the suffix "**/response /$request_id"** is applied

Example with topic **attribute**→ v1/devices/me/attributes/

v1/devices/me/attributes/response/$request_id

Where the $request_id parameter must have the same value as the one in the request.

The response message from the server consists of a JSON object containing the following key:

- **"shared."**
  - Type: JSON object
  - Description: contains the key-value pairs related to the keys required by the board.

The requested keys, excluding those related to firmware, are sent by the board with the prefix "s_". This ensures that if a parameter is changed while the board is off and the RPC request fails, the board can still receive the correct and updated values.

The possible keys required by the board are as follows:

- **s_energy_standby_threshold:**
  - Type: Integer
  - Description: instantaneous power threshold below which the board considers the machine in idle state in W. Refers to the parameter with the key sent by the board **"energy_standby_threshold"**.

- **s_inactivity_time:**
  - Type: Integer
  - Description: indicates after how long the CutOff relay is triggered when the machine is in idle state in seconds. It refers to the parameter with the key sent from the **"inactivity_time"** tab.

- **s_packet_time:**
  - Type: Integer
  - Description: field indicating how often the creation of a packet containing periodic data occurs in seconds. Refers to the parameter with the key sent from the **"packet_time"** tab.
- **fw_title:**
  - Type: String
  - Description: title Firmware OTA

- **fw_version:**
  - Type: String
  - Description: Firmware OTA version

- **fw_size:**
  - Type: Integer
  - Description: binary OTA file size [bytes]

- **fw_checksum:**
  - Type: String
  - Description: OTA firmware checksum calculated with MD5 algorithm

Sample server response:

Topic: v1/devices/me/attributes/response/1 QoS: 0


```
{
  "shared": {
    "fw_title": "TraceAndFollow",
    "fw_version": "1.0.2",
    "s_energy_standby_threshold": 3585,
    "s_inactivity_time": 95,
    "s_packet_time": 57
  }
}
```


OTA Updates

Through the MQTT interface, the firmware of the Trace&Follow board can be updated.

The update begins with the server posting the following keys in the **"attribute"** topic:

- **fw_title:**
  - Type: String
  - Description: title Firmware
  - Allowed values: **"TraceAndFollow"**, if the submitted name is different from the one given here, the package is not installed.

- **fw_version:**
  - Type: String
  - Description: Firmware version. The version must be greater than the previous version for the package to be installed.

- **fw_size:**
  - Type: Integer
  - Description: binary file size [bytes]

- **fw_checksum:**
  - Type: String
  - Description: firmware checksum calculated with MD5 algorithm

- **fw_tag:**
  - Type: String
  - Description: its value consists of fw_title and fw_version separated by a space.

- **fw_checksum_algorithm:**
  - Type: String
  - Description: algorithm used to calculate the checksum.
  - Allowed Values: **"MD5"** this field must have fixed value as reported

Example of a message published by the server:

Topic: v1/devices/me/attributes QoS: 0


```
{
  "fw_title": "TraceAndFollow",
  "fw_version": "1.0.2",
  "fw_tag": "TraceAndFollow 1.0.2",
  "fw_size": 1437520,
  "fw_checksum_algorithm": "MD5",
  "fw_checksum": "ad4d254d429695ad7cfbe2492192a7d4"
}
```


When the board receives valid parameters, it enters in Download mode and stops normal program execution. In this mode, it does not accept RPC calls or send data as it does during ordinary operation.
Once the board enters this mode, it posts a message on the topic **"attribute"** containing the following keys:

- **fw_chunk_size:** indicates the size of the chunks with which the server should respond to requests from the board
- **fw_chunk_number:** indicates the number of chunks that the board will request
- **fw_current_chunk:** indicates the chunk that is currently downloading. In this first message, its value will be 0

Example of first message:

Topic: v1/devices/me/telemetry QoS: 0


```
{
  "fw_chunk_size": 15000,
  "fw_chunk_number": 95,
  "fw_current_chunk": 0
}
```


Then the board posts another message on the same topic containing the status of the update:

Topic: v1/devices/me/telemetry QoS: 0


```
{
  "fw_state": "DOWNLOADING"
}
```

At this point the download of the binary file begins. The board public on the topic

v2/fw/request/0/chunk/$chunk_number

a message containing the size of the requested chunk. The value of $chunk_number is an incremental integer that corresponds to successive sections of the firmware.
The server will have to respond on the topic

v2/fw/response/0/chunk/$chunk_number

with a number of bytes of the binary file equal to the chunk_size. The value of the $chunk_number parameter should be the same as the value of the request made by the board.

Example of server response with $chunk_size = 100 and $chunk_number = 0

Topic: v2/fw/response/0/chunk/0 QoS: 0

```
e905 0220 2825 0840 ee00 0000 0000 0000 00ff ff00 0000 0001 2000 403f d4b9 0300 3254 cdab
0000 0000 0000 0000 0000 0000 7631 2e30 2e31 2d38 2d67 6438 6564 3430 342d 6469 7274 7900
0000 0000 0000 0000 5253 3232 3030 335f 4553 5033 325f 7632 0000 0000
```

When chunk 0 has been downloaded, the board publishes a request for the next chunk and so on until the end of the chunks. At each chunk, the board publishes the value of **"fw_current_chunk"**

Example of message published after receiving chunk 1:

Topic: v1/devices/me/telemetry QoS: 0

```
{
  "fw_current_chunk": 1
}
```

After the download is finished, the board notifies the success of the update by setting the value of "**fw_state**" to "DOWNLOADED"

Topic: v1/devices/me/telemetry QoS: 0

```
{
  "fw_state": "DOWNLOADED"
}
```

At this point the board reboots and installs the update.

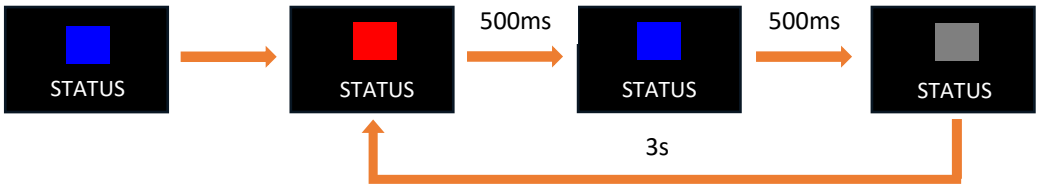
In case of an error during a step of the update, the parameter **"fw_state"** is set to "FAILED" and the parameter **"fw_error"** set with an agreement.

Example of an error where the firmware version is not greater than the one already present:

Topic: v1/devices/me/telemetry QoS: 0

```
{
  "fw_error": "Firmware is already up to date",
  "fw_state": "FAILED"
}
```

When the firmware update starts, the STATUS LED on the board turns BLUE.
In case of an error, the LED flashes red for 500ms, returns blue for another 500ms, after which it remains off for 3 seconds and the cycle is repeated twice.



I/O MANAGEMENT.

The Trace&Follow board has four physical inputs and four physical outputs. Additionally, it is configured to handle virtual inputs and outputs, which are used to standardize the handling of values sent from the board. Once the I/O is configured on the board, a standard set of keys on the MQTT interface can be used, regardless of the type of value being monitored.
Example: A pressure sensor is connected via the RS485 expansion port, returning values in bar.
The value is transmitted via MQTT using the virtual_input1_value key. In a similar application, the pressure sensor is substituted with an encoder. The board consistently sends the acquired value as virtual_input1_value, ensuring that the keys sent by the board remain the same, while the meaning of the value can be configured on the server-side.

Input configuration

The board inputs can be configured in 4 modes:

1. **Normal:** The input is considered normal digital input. A filter time can be set (key “filter_time” in the configuration file). State changes that occur within the set filter time will not be transmitted to the server. If the filter time is exceeded, the message will be transmitted via MQTT.

2. **Counter:** In this mode, the board counts incoming pulses on the pin and treats data for inputs configured in this mode as periodic data. The data is then inserted into packets with associated timestamps.

3. **Function:** In this mode, the Boolean variable inside the board changes value when the signal changes state. The variable starts at 0 during startup and changes to 1 when the signal changes state (edge). The variable can be reset to 0 if the signal is held high for at least 3 seconds. An example of an application could be a maintenance request button: the operator presses the button to bring the variable to 1, and to reset the state, they must press the same button for 3 seconds**.**

4. **Blinking:** This function detects a blinking input signal. For instance, it can detect a signal from a signal column of an industrial machinery. A flashing light corresponds to an active machine (value = 1), while a light off corresponds to a stopped machine (value = 0).

For each input, it is then also possible to configure the polarity of the input (logical inversion):

- "polarity" = 0: Normal polarity.

- "polarity" = 1: Reverse polarity.

Example: "polarity" = 0, in case of 0V applied on the input, the board sends 0 as status, in case of 24V applied, the board sends 1. If "polarity" = 1 on the other hand in case of 0V applied on the input, the board sends 1 as status, in case of 24V applied, the board sends 0.
When the input is configured in "Counter" mode, setting the "reset_counter" configuration key to "true" means that the counter value will be reset with every packet. On the other hand, if it is set to "false", the counter will not be reset up to a maximum value of 4,294,967,295 (32-bit unsigned integer).
When the input is configured in the other 3 modes, the "add_to_packet" key can be used to include or not include it as periodic data instead of sending it as asynchronous data. Specifically, if "add_to_packet" = "true", it is sent as periodic data, otherwise as asynchronous data.

Output configuration

Polarity can be configured for the outputs:

- "polarity" = 0: Normal polarity.
- "polarity" = 1: Reverse polarity.

Example: "polarity" = 0, in case of command to set the output to the value of 1, the output will be set to the value of VCC, in case of command to set the output to the value of 0, the output will be set to the value of GND.
"polarity" = 1, in case of command to set the output to the value of 1, the output will be set to the value of GND, in case of command to set the output to the value of 0, the output will be set to the value of VCC.

MODBUS TCP SERVER

The board is equipped with a Modbus TCP server for transmitting and receiving data via the Modbus TCP protocol.
Of the server it is possible to configure:

- Port on which the server is exposed.
- Slave id.
- The maximum number of simultaneously connected clients .
- The idle time before a client is disconnected (idle time is defined as the time for which the client does not send requests to the server).
- The activation or deactivation of the server.

These parameters are contained in the configuration file.

The Modbus TCP server can be used by an external PLC to enter production data related to current productions if the Trace&Follow board is used to monitor a production machine.
In this scenario, the PLC functions as a client and is responsible for establishing the connection to the server on the board.

The server and the board have the same IP address on the network. To access the board via its IP address, it must be connected via Ethernet or Wi-Fi.

If the board is not connected to a network or is only connected to 4G, you can still access the Modbus server by connecting to the access point generated by the board. The server will only be exposed on the same IP that is set by the configuration file on which the web page is exposed.

The commands to which the board responds are:

- Read Holding Registers (03)
- Read Input Registers (04)
- Write Multiple Registers (16)

The write command writes to the Holding Registers.

Input registers

Through the server, many parameters of the board, saved in the input registers according to the following table, can be accessed:

Add	meaning	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	notes
0	Fixed value at 1	1																
1	Hardware Version	major version								release								
2	Firmware Version	major v				minor v				release								
3	packet time	value in seconds																
4	transmission time	value in seconds																
5	inactivity time	value in seconds																
6	energy stand-by threshold	watts value																
7	connection type	1-> Ethernet, 2-> Wi-Fi, 3-> LoRa, 4 -> 4G																
8	input type	type in 4				type in 3				type in 2				type in 1				0-> disabled, 1-> normal, 2-> counter, 3-> function
9	AP enabled	1-> enabled, 0-> disabled																
10	AP name 1	character 29								character 28								Character 29 leftmost, character 0 rightmost. Unused characters set to 0
11	AP name 2	character 27								character 26								
12	AP name 3	character 25								character 24								
13	AP name 4	character 23								character 22								
14	AP name 5	character 21								character 20								
15	AP name 6	character 19								character 18								
16	AP name 7	character 17								character 16								
17	AP name 8	character 15								character 14								
18	AP name 9	character 13								character 12								
19	AP name 10	character 11								character 10								
20	AP name 11	character 9								character 8								
21	AP name 12	character 7								character 6								
22	AP name 13	character 5								character 4								
23	AP name 14	character 3								character 2								
24	AP name 15	character 1								character 0								
25	IP AP 1	byte 1 IP								byte 2 IP								byte4.byte3. byte2.byte1
26	IP AP 2	byte 3 IP								byte 4 IP								
27	cut-off enabled	1-> enabled, 0-> disabled																
28	localization enable status	1-> enabled, 0-> disabled																
29	reserved																	
30	network connection status	1->connected, 0->not connected																

31	MQTT connection status	1->connected, 0->not connected									
32	IP tab 1	byte 1 IP				byte 2 IP					byte4.byte3. byte2.byte1
33	IP tab 2	byte 3 IP				byte 4 IP					
34	Modbus client maximums										
35	Modbus clients connected										
36	Modbus idle timeout	value in seconds									
37	IO logical status	reserved	o4	o3	o2	o1	in4	in3	in2	in1	possible values for each field 0/1
38	instantaneous power	watts value									
39	reserved										
40	RPC 0	Set by RPC call									Sent to cloud with add 3000
41	RPC 1	Set by RPC call									Sent to cloud with add 3001
42	RPC 2	Set by RPC call									Sent to cloud with add 3002
43	RPC 3	Set by RPC call									Sent to cloud with add 3003
44	RPC 4	Set by RPC call									Sent to cloud with add 3004
45	RPC 5	Set by RPC call									Sent to cloud with add 3005
46	RPC 6	Set by RPC call									Sent to cloud with add 3006
47	RPC 7	Set by RPC call									Sent to cloud with add 3007
48	RPC 8	Set by RPC call									Sent to cloud with add 3008
49	RPC 9	Set by RPC call									Sent to cloud with add 3009

The RPC registers 0-9 can have their values set via the MQTT interface using the **“setInputRegister”** method provided in the RPC method table. This allows for server-sent values to be read by the PLC connected to the Trace&Follow board. Additionally, the configuration file includes a “send_input” key in the “MODBUS_TCP” field. When the key is set to "true", when the value is written via RPC, the board sends the value just set on the **"telemetry"** topic with the key **"virtual_input300x_value".**

Example of incoming request from server with board response when "send_input" = "true":

Holding registers

Unlike Input Registers, Holding Registers can be written by the PLC. They are used to monitor the output of the machine to which the board is connected.
The logs are divided into 5 blocks, allowing up to 5 simultaneous productions to be monitored.
For each production, you can set:

- The production **code** (Max 10 characters): registers x0 - x4, 2 ASCII characters for each register.
- The **OK** piece counter: register x5
- The **NOK** piece counter: register x6
- The **cycle time of** production: log x7
- The production **target**: register x8

Finally, there is a register through which the PLC can send the status of the machine to the server.

This is data that the PLC writes, and the board sends to the server via the MQTT interface.

The board publishes this data as soon as it is written using the **"virtual_input(N)_value"** keys according to the following table:

Address	Meaning	Virtual Input Value	Production
1	machine status	4001	/
10	product code 1	4014	**PRODUCTION 1**
11			
12			
13			
14			
15	piece counter OK 1	4015	
16	scrap piece counter 1	4016	
17	cycle time 1	4017	
18	target production 1	4018	
20	product code 2	4024	**PRODUCTION 2**
21			

22			
23			
24			
25	piece counter OK 2	4025	
26	scrap piece counter 2	4026	
27	cycle time 2	4027	
28	target production 2	4028	
30	product code 3	4034	**PRODUCTION 3**
31			
32			
33			
34			
35	piece counter OK 3	4035	
36	scrap piece counter 3	4036	
37	cycle time 3	4037	
38	target production 3	4038	
40	product code 4	4044	**PRODUCTION 4**
41			
42			
43			
44			
45	piece counter OK 4	4045	
46	scrap piece counter 4	4046	
47	cycle time 4	4047	
48	target production 4	4048	
50	product code 5	4054	**PRODUCTION 5**
51			
52			
53			
54			
55	piece counter OK 5	4055	
56	scrap piece counter 5	4056	
57	cycle time 5	4057	
58	target production 5	4058	

At the start of a new production, the PLC must set registers x0 - x4 with the corresponding production code. The board will then send the code as a single string using the key “virtual_input_40x4_value” Additionally, cycle time and target can be set and sent in a similar manner.

During the production phase, the PLC can set the registers associated with the counters according to the production statistics. When a register is set to a non-zero value, the board sends that value to the server and resets the register to 0 to indicate successful transmission.
After production is complete, the PLC should reset the production code, cycle time, and target registers to 0. The board sends the reset parameters and a null string as the code to indicate the end of production.

Example production start with only target and code set:

Topic: v1/devices/me/telemetry QoS: 0

```
[
  {
    "virtual_input4014_value": "GreenPen1",
    "virtual_input4018_value": 576
  }
]
```

Example writing counters:

Topic: v1/devices/me/telemetry QoS: 0

```
[
  {
    "virtual_input4015_value": 16,
    "virtual_input4016_value": 8
  }
]
```

It is also possible to set only one counters. The board sends only those that have been changed:

Topic: v1/devices/me/telemetry QoS: 0

```
[
  {
    "virtual_input4015_value": 6
  }
]
```

Example end of production:

Topic: v1/devices/me/telemetry QoS: 0

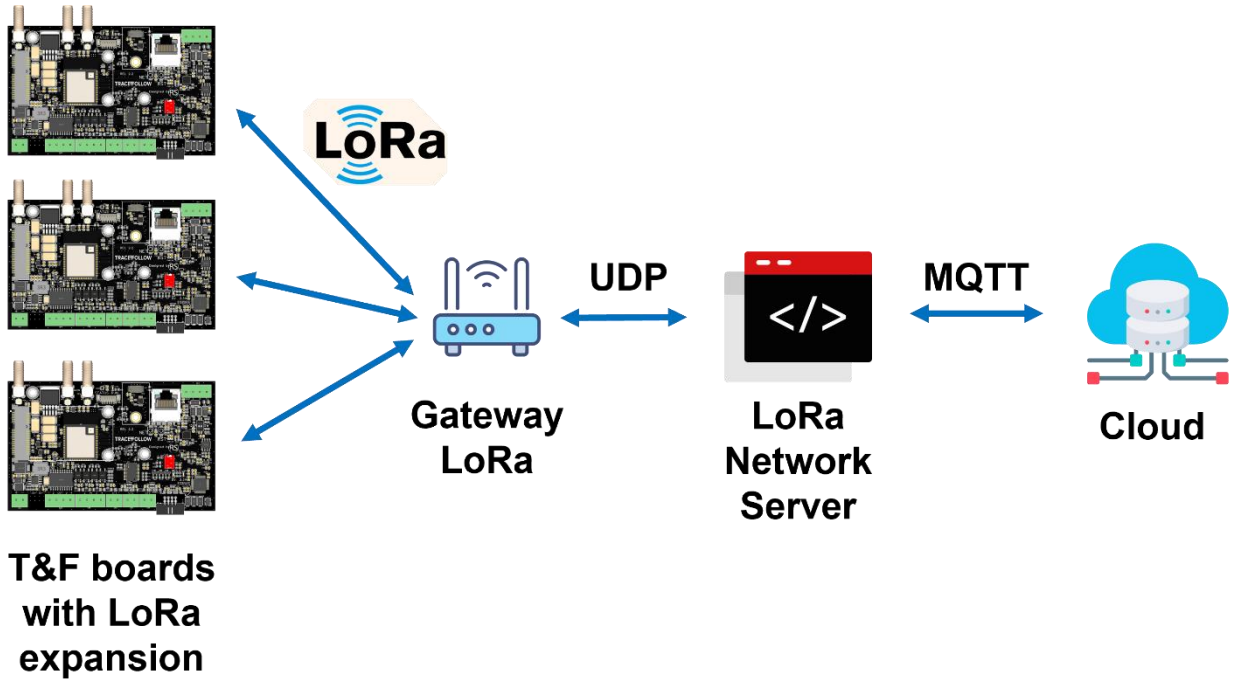
```
[
  {
    "virtual_input4014_value": "",
    "virtual_input4018_value": 0
  }
]
```

LoRaWAN INTERFACE

With the LoRaWAN extension, the Trace&Follow board can be configured to work over the LoRaWAN protocol. This protocol allows sending and receiving data over long distances via a LoRa gateway in environments where there is no connectivity (the gateway must still be connected to the network).
The configuration file can be used to set the connection parameters to the gateway.
The connection is established using the class A Over-The-Air-Activation (OTAA) procedure.
The communication parameters are:

- LoRaWAN MAC version: 1.0.2
- LoRaWAN Regional Parameters revision: B
- ADR algorithm: Default ADR algorithm (LoRa only)
- Max EIRP: 27
- Uplink Interval: 1

LoRa infrastructure scheme:



Trace&Follow cards send over LoRa packets encoded in a byte array, which you can decode using the function attached to this document.
You can also send data to the card via this protocol using the encoding function attached to this document or downloadable from https://idtsolution.com/trace-and-follow/.
Both functions must be implemented on the LoRa network server. The decoding function returns a JSON, while the encoding function expects a JSON as input. Both are formatted as described in the MQTT connection section.

Note: Parameters related to qualitative energy measurement are not available through this protocol, nor is the interface to the Modbus TCP server. OTA updates are also not available.

REVIEWS

REVIEWS		
N.	**Description**	**Date**
0	First Release	04/04/2024
1	Introduction qualitative energy data, LoRaWAN connection	11/06/2024
2	Update measurement unit for sent data	18/06/2024
3	Update pga_gain values for energy meter calibration	29/07/2024

This document constitutes technical documentation; for further details and information, please refer to the full Trace&Follow™ manual.