LucidControl


User Manual

# LucidIoT

## 16 Channel Secure Network Data Acquisition and Control Unit

# Table of Content

# 1 Introduction

LucidIoT is a secure low-power secure network data acquisition and control device for IoT applications. LucidIoT provides many functions like data logging and supports multiple communication protocols.

LucidIoT is the next generation of the LucidControl USB IO modules, which are used by the automation industry since many years. LucidIoT brings their functionality to the IoT world.

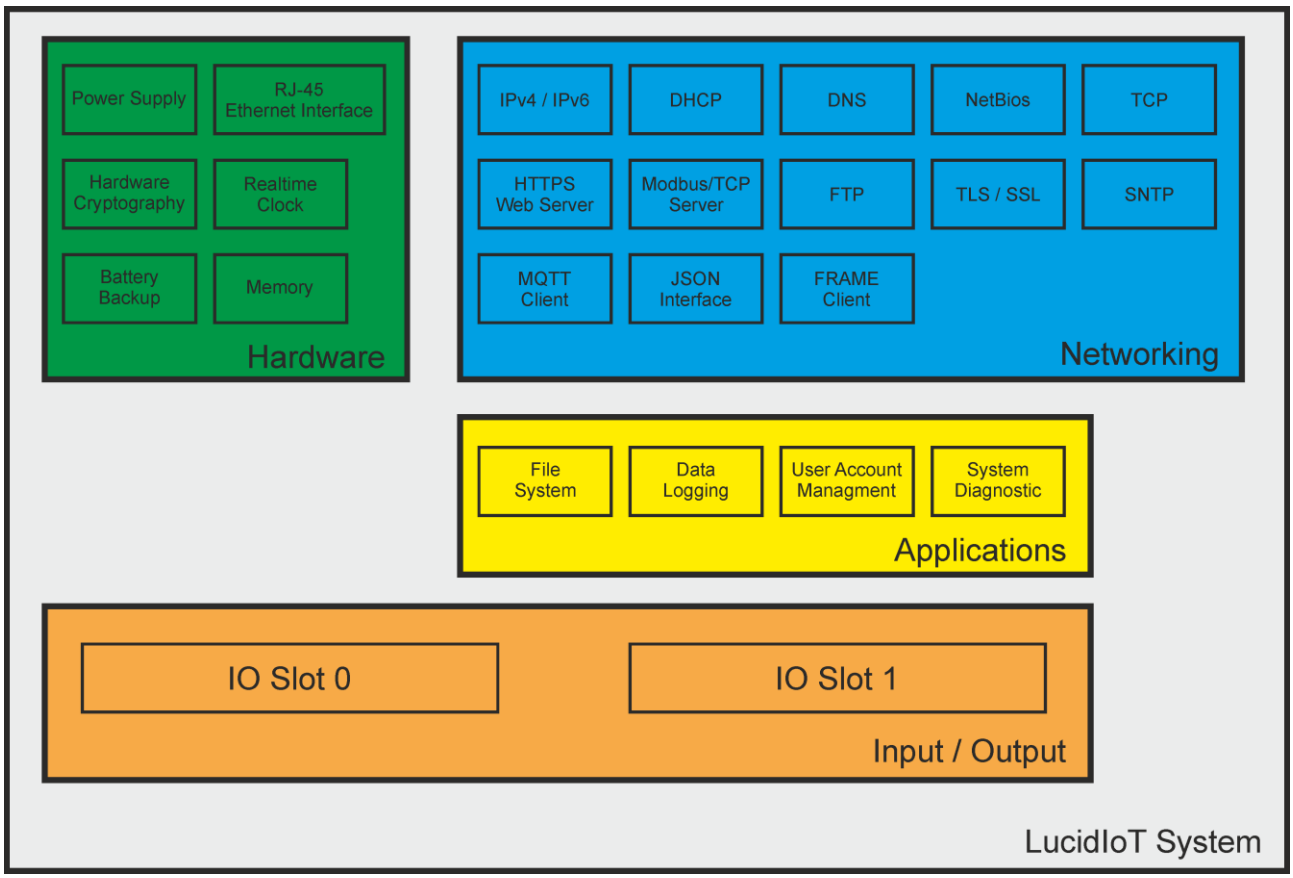



**Fig. 1 LucidIoT System Schematic**

Fig. 1 shows the function blocks of the LucidIoT.

LucidIoT is a customizable data acquisition and control solution providing up the 16 IO channels.

LucidIoT can be mounted in switch cabinets and the housing can be fitted on DIN-Rails. It is power supplied by a 9-24V source. Optional galvanic isolation is available.

LucidIoT allows installation of two IO slots 0 and 1. Each IO slot provides its own function class and type. A function class is e.g. digital output or analog input and defines the general functionality of the IO slot. The function type specifies details of the function class like input voltage range (→ 4).

LucidIoT multi-protocol IO controller communicates via Ethernet 10 Mbit / 100 Mbit. It supports IPv4 and IPv6 networks.

LucidIoT communication interfaces:

- Basic protocols IPv4, IPv6, TCP, UDP, DHCP, DNS, NetBIOS
- Modern HTTPS web server optimized for mobile devices (responsive)
    - Customized web pages can be loaded to LucidIoT
- Modbus/TCP
- MQTT client (Amazon AWS, Google IoT, Mosquitto)
- FTP
- JSON and LucidIo FRAME protocol
- SNTP synchronized real time clock
- Protocols can be optionally TLS secured with X.509 certificates

LucidIoT provides the following functions and services:

- File system accessible by FTP
- Data logging functionality
- Real Time Clock with battery backup
- Strong hardware cryptography
- User account access management
- System diagnostic functions
- Firmware update function

The IoT controller optionally secures the network protocols (e.g. HTTP web server or MQTT client) by TLS and X.509 certificates. In addition, the user account management provides functions to restrict user access rights.

LucidIoT is able to log data of the IO slots as well as system diagnostic values. Data is stored in the internal file system and can be downloaded by FTP.

LucidIoT devices can operate with the LucidControl standard tools. The IO modules implement the FRAME protocol (→ 5.4), which is compatible with LucidControl USB IO modules. This makes the migration to LucidIoT easy and allows the user to get a quick start e.g. by using the LucidIoCtrl command line tool.

APIs made for LucidIoT are available for .NET and Python. But, since the protocol is fully documented, the user can develop his own interface to LucidIoT very easily, e.g. using the JSON interface (→ 5.3).

# 2 Setup and Installation

## 2.1 Hardware Installation

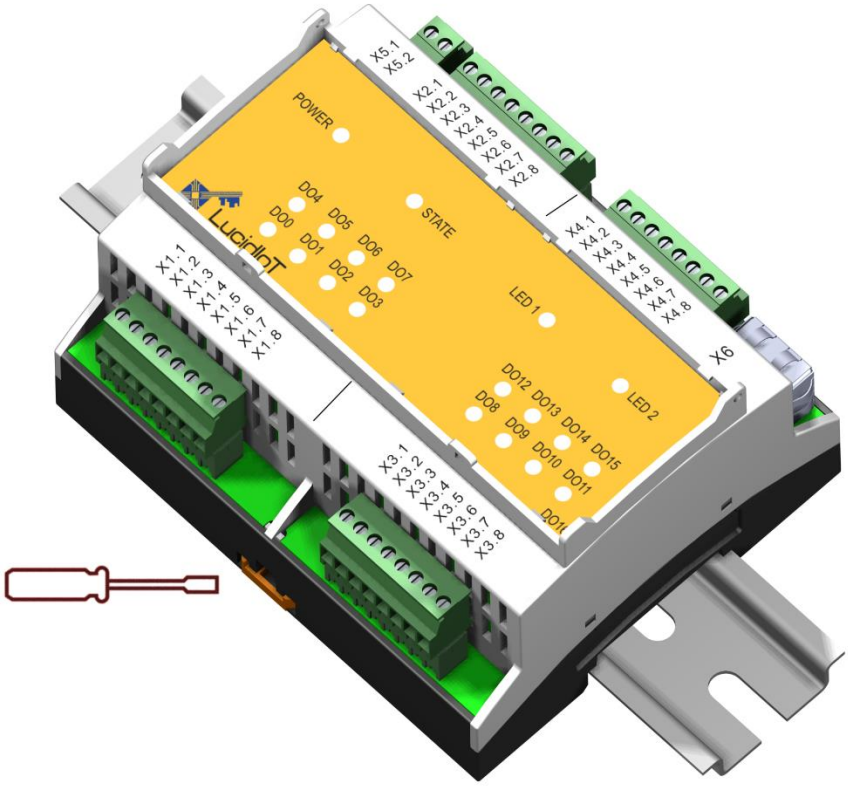



**Fig. 2 LucidIoT on DIN-Rail**

To attach the LucidIoT enclosure on the DIN-Rail, tilt and press the housing onto the fixed rail. The two orange fixtures will clip into the upper and lower rim of the rail holding the module firmly.

The module can be released from the DIN-Rail by bending the two orange fixtures outwards by using a skew driver (Fig. 2).

Please see the following sections for electrical wiring:

- Power supply (→ 3.1)
- Network connection (→ 3.2.1)
- IO Signal connection
    - Digital input function class (→ 4.2.1.1)
    - Digital output function class (→ 4.2.2.1)
    - Analog input function class (→ 4.2.3.1)
    - Analog output function class (→ 4.2.4.1)
    - RTD input function class (→ 4.2.5.1)

## 2.2 Start of LucidIoT

When LucidIoT is powered-up, the green POWER LED blinks in 1s intervals until the device is ready.

Some services may need longer time during start and some may fail. If the start of a critical service fails, this is indicated by the active red STATE LED. LucidIoT should not be used in this case.

In normal operation, the green POWER LED is active.

When LucidIoT has started, the IO processing is running, and a user can login as described in section 3.3.9.2.

### 2.2.1 Troubleshooting

When the start of LucidIoT fails, this can have the following reasons:

- DHCP service failed, server did not respond with IP address
- File system could not be initialized

If DHCP is enabled, LucidIoT is only functional when and IP address was assigned. If the DHCP server had not assigned an IP address after a timeout (of e.g. 30s), the start of the networking service failed. In this case, the device can be started in maintenance mode (→ 3.4.7). Restoring the factory default configuration can solve the situation.

The file system is initialized on start. If this fails the device is operational, but services requiring it (e.g. data logging and system diagnostic) are not working. Most likely, the internal memory needs to be formatted in maintenance mode (→ 3.4.7) to solve this problem.

## 2.3 Safety information

LucidIoT complies with regulations and industrial standards active in the EU. To keep the device functional, the following safety and maintenance information must be adhered.

LucidIoT must only be used for the intended purpose.

LucidIoT must not be used under the following conditions:

- It is obviously damaged
- An error was detected
- Outside humidity and temperature limits
- Unauthorized personnel

Never apply voltages higher than 30V (or lower than -30V) to any IO terminal. This would damage the device.

# 3 Functional Description

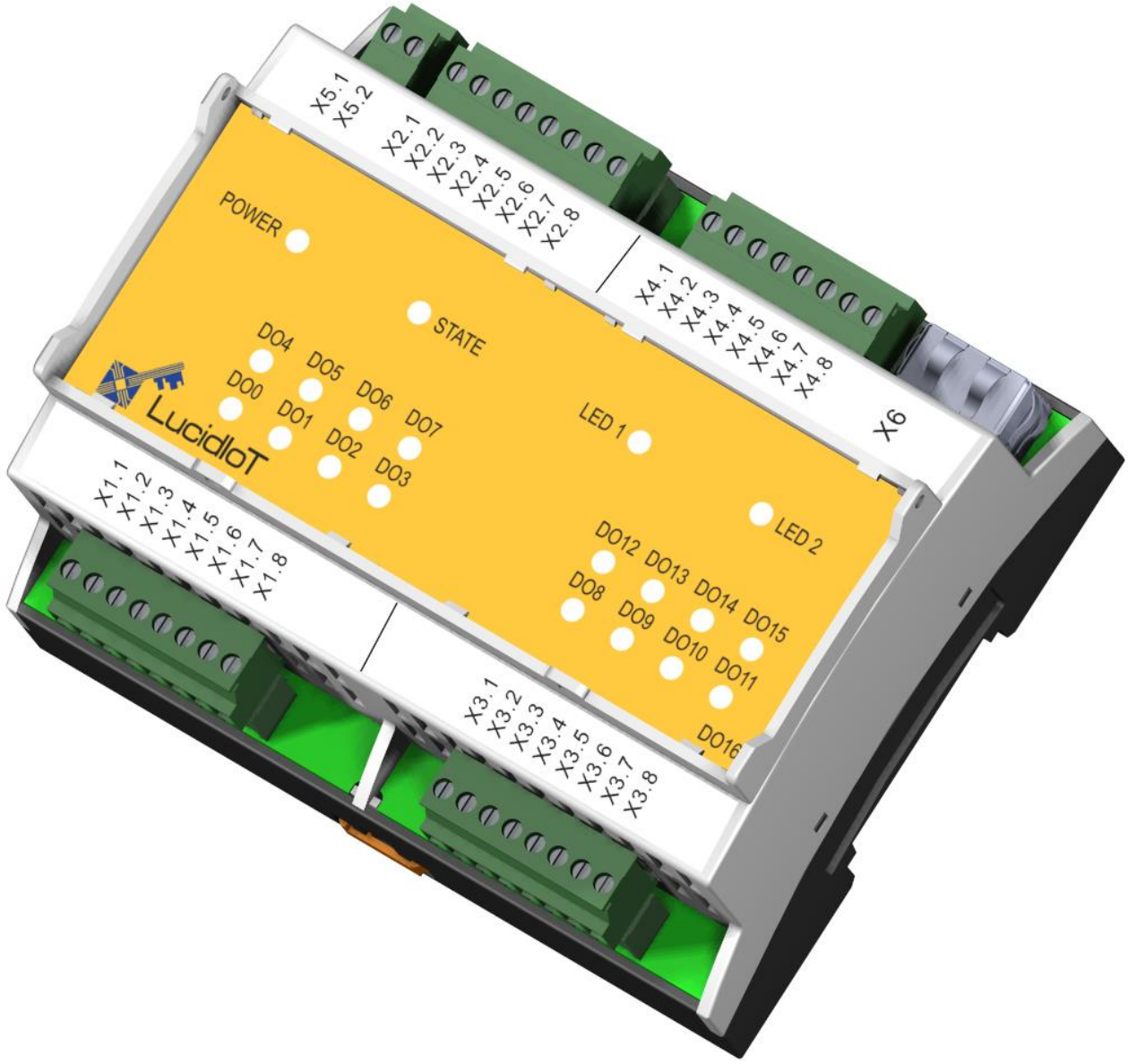



**Fig. 3 LucidIoT Model**

## 3.1 Power Supply

LucidIoT is powered by a power supply connected to X5 (Tab. 1).

| Connector | Terminal | |
|---|---|---|
| X5 | 1 | +9 ~ +24V positive power supply |
| | 2 | 0V, Ground |

**Tab. 1 Power Supply Connection**

In the standard version, X5.2 is connected to internal ground.

## 3.2 Ethernet Interface

### 3.2.1 Network Links

The RJ45 network cable is connected to X6.

| LED | Function |
|---|---|
| Green | Switched constant on when device is connected to Ethernet and a link was established |
| Orange | Network activity blinking |

**Tab. 2 X6 LED Function**

Tab. 2 explains the functionality of the LEDs of the Ethernet connector.

## 3.3 Network Services

### 3.3.1 Ethernet and TCP/IP

This section describes how the LucidIoT Ethernet and TCP/IP service works and how to configure it.

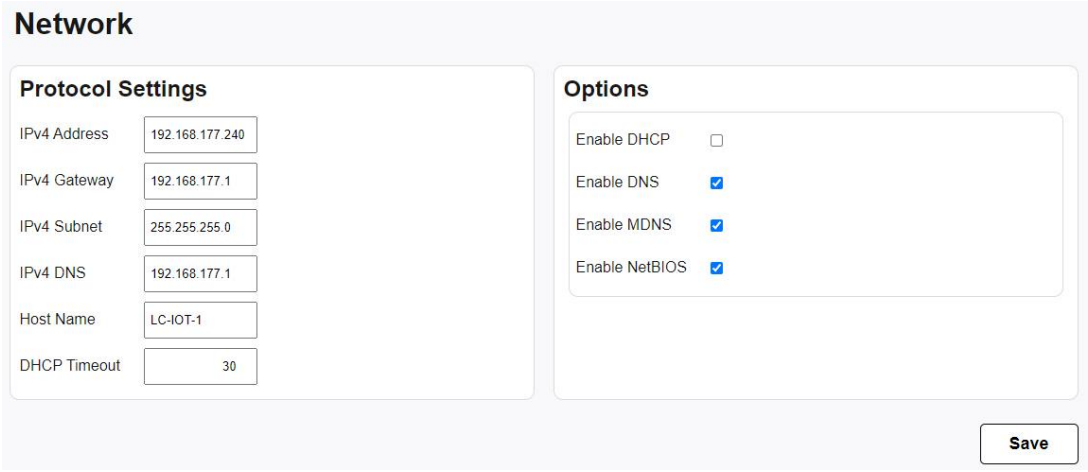



**Fig. 4 Network Configuration Page**

Fig. 4 shows the upper part of the network configuration page.

Only users with admin rights are allowed to access this page. It is not visible for other users.

Protocol Settings, Options and Link Support

| Parameter | |
|---|---|
| IPv4 Address | IPv4 address |
| IPv4 Gateway | IPv4 gateway |
| IPv4 Subnet | IPv4 subnet |
| IPv4 DNS | IPv4 address of the name server |
| Host Name | Host name |
| DHCP Timeout | DHCP timeout in seconds<br>If DHCP is enabled and the server does not reply with an IP address, error is signaled.<br>Value 0 disables timeout and waits forever |
| Enable DHCP | Enables DHCP |
| Enable DNS | Enable DNS |
| Enable MDNS | Enable Multicast DNS |
| Enable NetBIOS | Enable NetBIOS |

**Tab. 3 Network Configuration Values**

The configuration is updated by clicking the *Save* button. Updating parameters may require a restart of the device. In this case, the current configuration remains active and the Service State flag *Reset* is set to true. The new configuration becomes active after a restart.

### 3.3.2 Physical Layer

LucidIoT is compatible with IEEE 802.3 standard and operates with 10BASE-T and 100BASE-T links. The optimum link is selected by auto-negotiation.

### 3.3.3 IP Communication

LucidIoT can be configured to operate in IPv4 and/or IPv6 networks.

The parameters *IPv4 Address*, *IPv4 Gateway*, *IPv4 Subnet* and *IPv4 DNS* specify the addresses of the IPv4 protocol.

The default IPv4 address is 192.168.177.240.
In maintenance mode (→ 3.4.7) the default IPv4 address is used.

### 3.3.4 DHCP

If Dynamic Host Configuration Protocol (DHCP) is enabled, LucidIoT requests an IP address from a DHCP server. The assigned address can be either static or dynamic depending on the server setup.

If DHCP is enabled by *Enable DHCP* configuration parameter, LucidIoT network operates only if an IPv4 address was successfully assigned to the device. If the DHCP request failed, or has not completed after *DHCP Timeout*, the device becomes unreachable and error state is signaled.

In a typical data acquisition and control application, it must be well considered if DHCP is required and which benefits are expected from using it. In general, we do not recommend enabling DHCP but using static IP addresses instead.

### 3.3.5 DNS

The Domain Name System (DNS) allows LucidIoT (e.g. the MQTT client) to resolve a domain name to its address.

The option *Enable DNS* must be set in the network configuration and the specific service configuration in order to become effective.

## 3.3.6 MDNS and NetBios

The Multicast DNS (MDNS) and NetBios protocols are used by LucidIoT in order to resolve the device name configured in the parameter *Host Name*.

The protocols can be activated by *Enable MDNS* and *Enable NetBIOS* parameters.

## 3.3.7 TCP

### 3.3.7.1 TCP Keep Alive

When a TCP connection is established, LucidIoT allocates resources required by this connection. The resources are released, when the connection is closed.

In the case that a connection is not shut down properly (e.g. because of crash or loss of networking connection), the connection remains open because of unawareness. Such half-open connections may lead to problems because of resource constraints.

LucidIoT uses the TCP keep alive function in order to detect half-open connections and closes them after a configurable timeout. The timeout can be configured separately for each module by the *Keep Alive* parameter.

By default, the keep alive timeout is set to 7200 s. This is the minimum value defined in RFC 1122. A value of 0 disables the TCP keep alive function.

## 3.3.8 Network State and Statistics

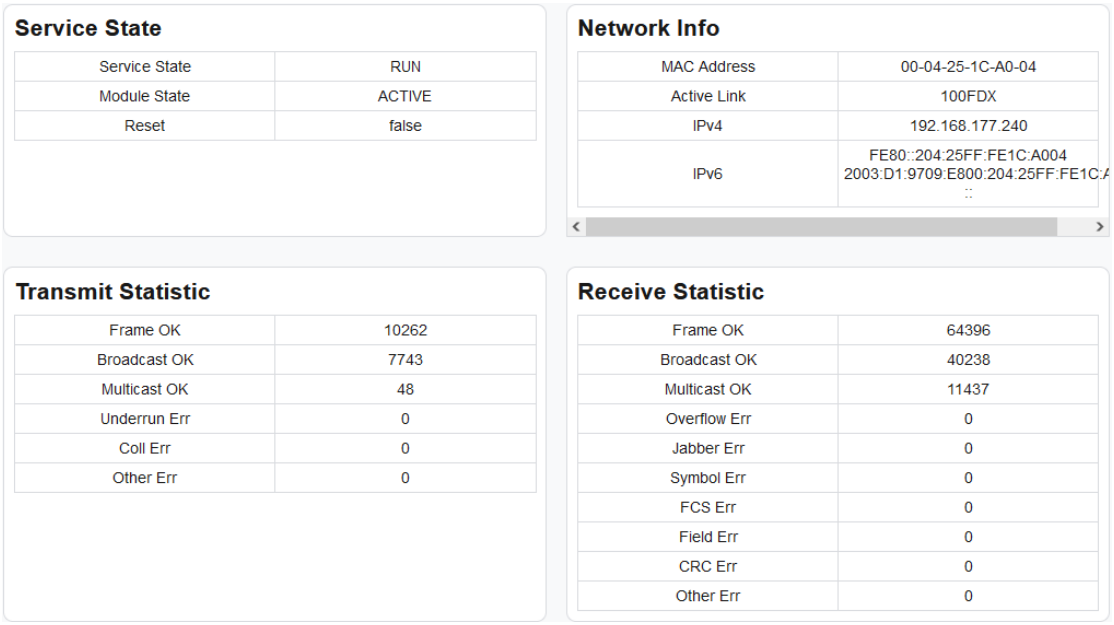



**Fig. 5 Network State and Statistic**

Fig. 5 shows the lower section of the network protocol configuration page.

The section *Service State* provides information of the network service.
The *Reset* value indicates if it is necessary to restart LucidIoT in order to apply configuration updates.
See also section Service and Module State in System Status Overview (→3.4.1, Fig. 22).

The section *Network Info* informs about the *MAC Address*, the *Active Link* (10HDX, 10FDX, 100HDX, 100FDX) and the IPv4 and IPv6 addresses.

The *Transmit Statistic* section lists statistic values of the Ethernet transmission. The values are reset daily after saving daily diagnostic information (→ 3.4.9.2).

| Value | |
|---|---|
| Frame OK | Number of successfully transmitted frames by the MAC. |
| Broadcast OK | Number of successfully transmitted broadcast frames. |
| Multicast OK | Number of successfully transmitted multicast frames. |
| Underrun Err | Number of MAC transmission buffer underruns. |
| Coll Err | Number of frames transmission failed because of collision. |
| Other Err | Number of frames transmission failed because of other reasons. |

**Tab. 4 MAC Transmission Statistic**

The *Receive Statistic* section lists statistic values of the Ethernet receive path. The values are reset daily after saving the daily diagnostic information (→ 3.4.9.2).

| Value | |
|---|---|
| Frame OK | Number of successfully received frames by the MAC. |
| Broadcast OK | Number of successfully received broadcast frames. |
| Multicast OK | Number of successfully received multicast frames. |
| Overflow Err | Number of MAC reception buffer overflows. |
| Jabber Err | Number of received frames with Jabber errors. |
| Symbol Err | Number of received frames with Symbol errors. |
| FCS Err | Number of received frames with Frame Check Sequence errors. |
| Field Err | Number of received frames with Field Frame errors. |
| CRC Err | Number of received frames with CRC errors. |
| Other Err | Number of received frames with other errors. |

**Tab. 5 MAC Receive Statistic**

## 3.3.9 HTTP Server

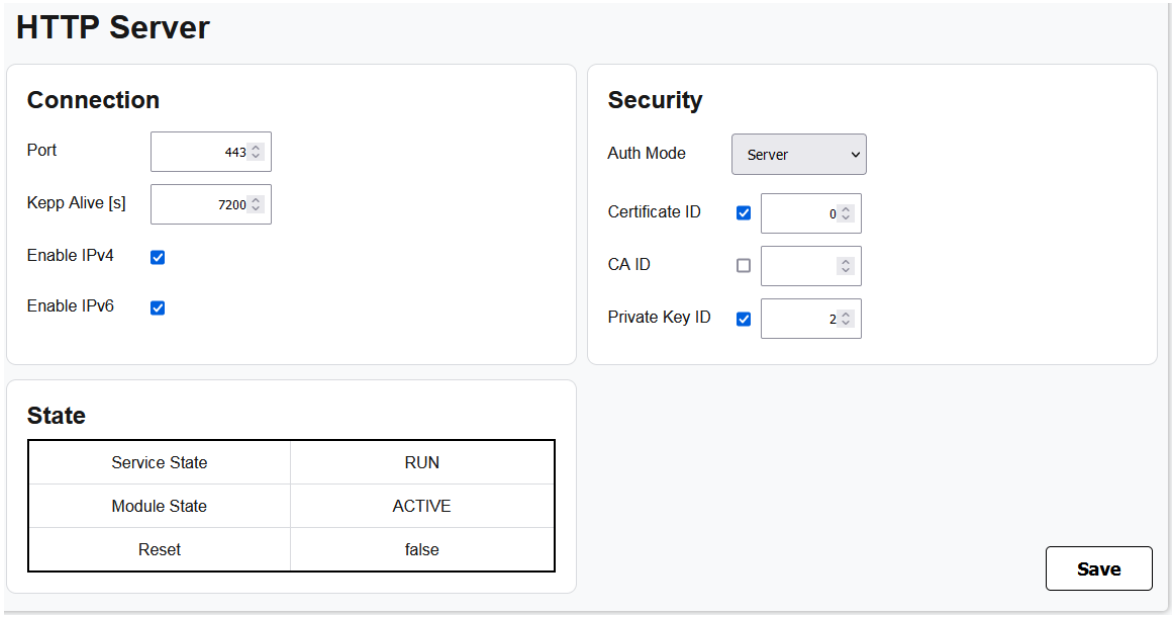



**Fig. 6 HTTP Server Configuration and State Page**

The HTTP server configuration and state page gives access to the configuration of the HTTP web server. Fig. 6 shows a typical configuration with for HTTPS.

Only users with admin rights are allowed to access this page. It is not visible for other users.

Connection Configuration

| Parameter | |
|---|---|
| Port | Port number LucidIoT is listening to<br>Default values are 80 for HTTP and 443 for HTTPS. |
| Keep Alive | Keep Alive Time in seconds (→ 3.3.7.1) |
| Enable IPv4 | Enable IPv4 address for HTTP server |
| Enable IPv6 | Enable IPv6 address for HTTP server |

**Tab. 6 HTTP Server Configuraton**

Security Configuration

| Parameter | |
|---|---|
| Auth Mode | HTTP server authentication mode<br>If set to Disable, HTTP is used.<br>If set to Server, TLS secured HTTPS is used. The HTTP server sends the certificate when a client connects |
| Certificate ID | If checkbox is enabled, the value contains the identifier of certificate in secure memory (→ 3.4.6.1).<br>If the checkbox is disabled, no certificate is provided. |
| CA ID | If the checkbox is enabled, the values contain the identifier of the CA or intermediate certificate in secure memory (→ 3.4.6.1).<br>If the checkbox is disabled, no certificate is provided. |
| Private Key ID | If the checkbox is enabled, the value containt the identifier of the private key in secure memory (→ 3.4.6.1).<br>If the checkbox is disabled, no private key is provided. |

**Tab. 7 HTTP Server Security Configuration**

Settings of *Certificate ID*, *CA ID*, *Private Key ID* are only relevant if *Auth Mode* is set to Server.

The configuration is updated by clicking the *Save* button. Updating parameters may require a restart of the device. In this case, the current configuration remains and the Service State flag *Reset* is set to true. The new configuration becomes active after a restart.

#### 3.3.9.1 Web Interface

LucidIoT can be accessed by a web browser. The web interface communicates with LucidIoT by using JSON protocol frames (→ 5).

#### 3.3.9.2 Login

A user can login to the device by the addresses:

http://192.168.177.240/login.htm or
https://192.168.177.240/login.htm

Note:
The default IP address is 192.168.177.240. It is used by a new device or if LucidIoT is operating in maintenance mode (→ 3.4.7).

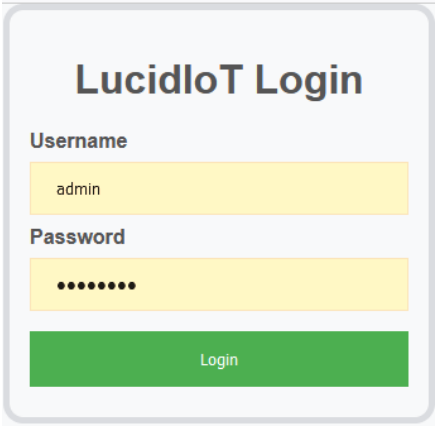



**Fig. 7 LucidIoT Webserver Login**

When the URL was opened, the login page appears requesting user login data.

A new device has the Master Admin Account enabled only (→ 3.4.5.1).

When the user logged in properly, the *System Information* page (→ 3.4.1) is loaded.

The accessible sub-pages and the navigation depend on the users' access rights (→ 3.4.5).

See 3.4.7 if LucidIoT operates in maintenance mode.

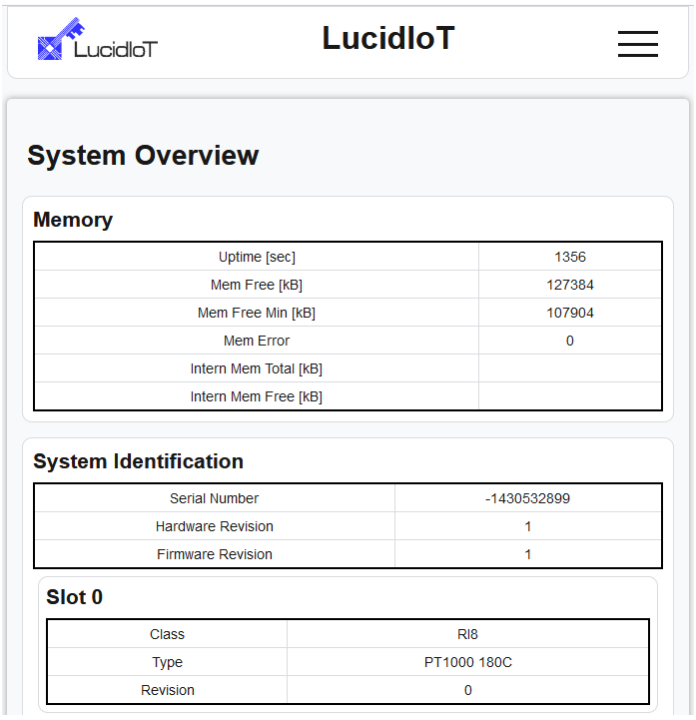
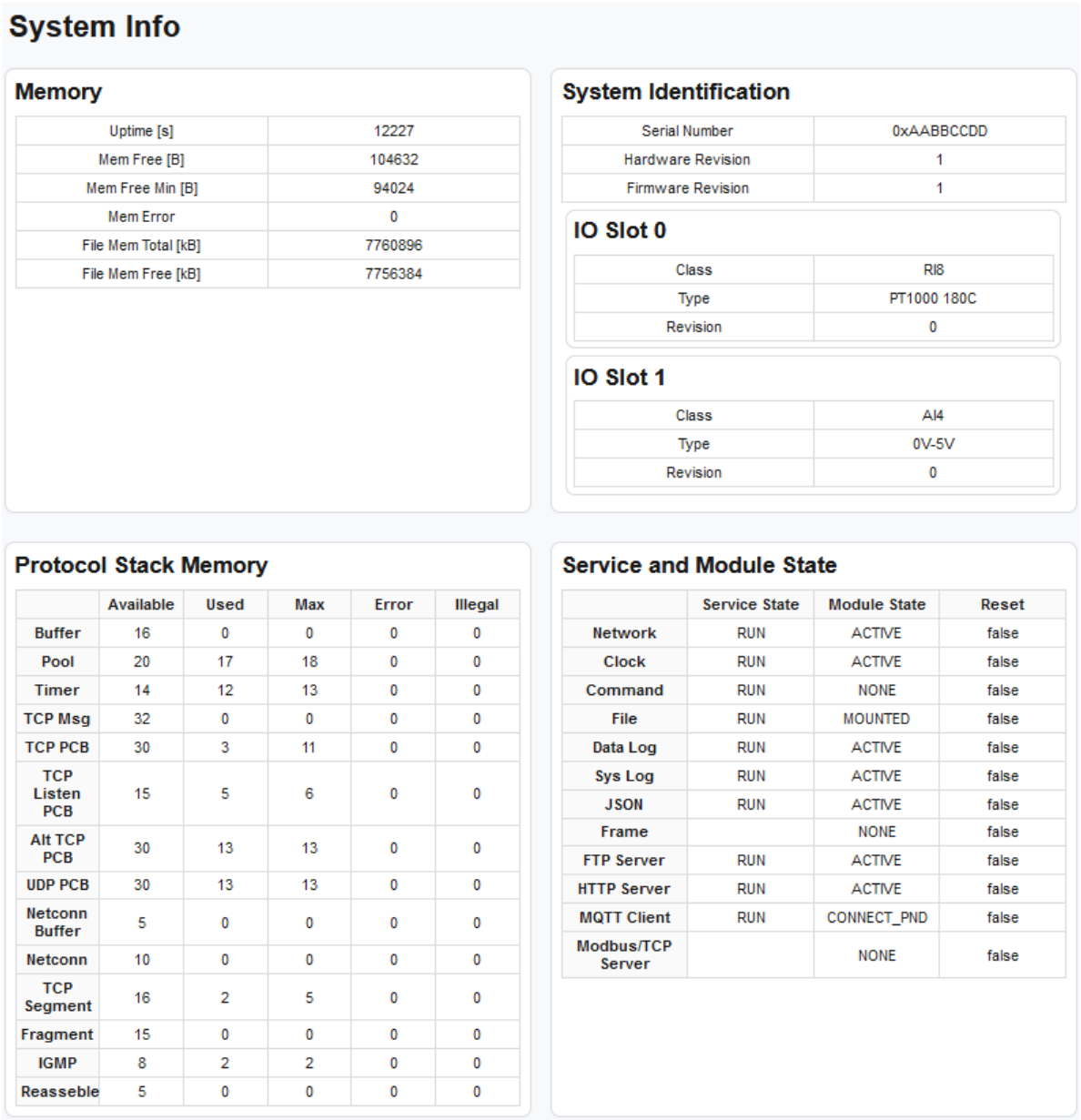



**Fig. 8 System Information (Responsive)**

The webpage uses a responsive design and the layout is optimized for the screen resolution of a client.

Fig. 8 shows how the responsive *System Information* page looks like on a mobile device.



System Info

Memory

| | |
|---|---|
| Uptime [s] | 12227 |
| Mem Free [B] | 104632 |
| Mem Free Min [B] | 94024 |
| Mem Error | 0 |
| File Mem Total [kB] | 7760896 |
| File Mem Free [kB] | 7756384 |

System Identification

| | |
|---|---|
| Serial Number | 0xAABBCCDD |
| Hardware Revision | 1 |
| Firmware Revision | 1 |

IO Slot 0

| | |
|---|---|
| Class | RI8 |
| Type | PT1000 180C |
| Revision | 0 |

IO Slot 1

| | |
|---|---|
| Class | AI4 |
| Type | 0V-5V |
| Revision | 0 |

Protocol Stack Memory

| | Available | Used | Max | Error | Illegal |
|---|---|---|---|---|---|
| Buffer | 16 | 0 | 0 | 0 | 0 |
| Pool | 20 | 17 | 18 | 0 | 0 |
| Timer | 14 | 12 | 13 | 0 | 0 |
| TCP Msg | 32 | 0 | 0 | 0 | 0 |
| TCP PCB | 30 | 3 | 11 | 0 | 0 |
| TCP Listen PCB | 15 | 5 | 6 | 0 | 0 |
| Alt TCP PCB | 30 | 13 | 13 | 0 | 0 |
| UDP PCB | 30 | 13 | 13 | 0 | 0 |
| Netconn Buffer | 5 | 0 | 0 | 0 | 0 |
| Netconn | 10 | 0 | 0 | 0 | 0 |
| TCP Segment | 16 | 2 | 5 | 0 | 0 |
| Fragment | 15 | 0 | 0 | 0 | 0 |
| IGMP | 8 | 2 | 2 | 0 | 0 |
| Reasseble | 5 | 0 | 0 | 0 | 0 |

Service and Module State

| | Service State | Module State | Reset |
|---|---|---|---|
| Network | RUN | ACTIVE | false |
| Clock | RUN | ACTIVE | false |
| Command | RUN | NONE | false |
| File | RUN | MOUNTED | false |
| Data Log | RUN | ACTIVE | false |
| Sys Log | RUN | ACTIVE | false |
| JSON | RUN | ACTIVE | false |
| Frame | | NONE | false |
| FTP Server | RUN | ACTIVE | false |
| HTTP Server | RUN | ACTIVE | false |
| MQTT Client | RUN | CONNECT_PND | false |
| Modbus/TCP Server | | NONE | false |

**Fig. 9 System Information**

Fig. 9 shows the system overview page, which is the first visible page after a login. The page shows the full navigation menu for a user having master admin access rights. For user with restricted access, the accessible navigation elements are visible only.

#### 3.3.9.3 Customized Web Content

Customized web content can be stored in the file system.
Please contact us if you need a customized web page, e.g. an user interface, presenting measurement data in a specific layout.

## 3.3.10 MQTT Client

The MQTT client provides connectivity to MQTT servers (brokers). It can exchange data by publishing and subscribing topics.

The MQTT client is compatible with MQTT 3.1.1 specification.

MQTT brokers have different procedures how a client can connect and authenticate.

LucidIoT supports TLS secured connections and can be configured for major servers like Eclipse Mosquitto, Amazon AWS IoT or Microsoft Azure.

MQTT client is disabled by default.

Data accessible by MQTT:

- IO channel values (e.g. measured voltages)
- IO channel configuration parameters (e.g. period and duty-cycle of digital outputs)
- Device Status (e.g. errors and system time)

Data of published topics are sent to the MQTT server, subscribed topics are received from a MQTT sever.

Published or subscribed topics are linked to LucidIoT values by *Value Identifiers* (→ 3.3.10.4.1). The data is formatted according to the value format configuration (→ 3.3.10.4.4).

The payload of published or subscribed topics contains (exact) one formatted IO Value if value identifiers of Tab. 13 are used

The payload of published or subscribed topics may contain multiple JSON formatted values if a Value Group identifier of Tab. 14 is used.

The MQTT client has full access to LucidIoT value identifiers. The user account management (→ 3.4.5) is disabled for this service.

#### 3.3.10.1 Endpoints and Server Name Indication (SNI) Extension

When LucidIoT connects to a secure MQTT server, the server identifies itself with its certificate. A single server can offer multiple virtual MQTT servers, all running at the same IP address. LucidIoT includes the SNI extension in its *Client Hello* TLS handshake message.

Some MQTT servers (e.g. Amazon AWS IoT) require presence of SNI Extension.

#### 3.3.10.2 Last Will and Testament (LWT)

Last will and testament function notifies other MQTT clients about an ungracefully disconnected client. LucidIoT MQTT client can specify a LWT message and topic, Quality of Service (QoS) and payload.

LWT is enabled if *LWT Topic* and *LWT Message* are defined.

The MQTT server publishes the message to all clients in case that LucidIoT disconnected unexpectedly and informs them about the offline status.

### 3.3.10.3 Configuration

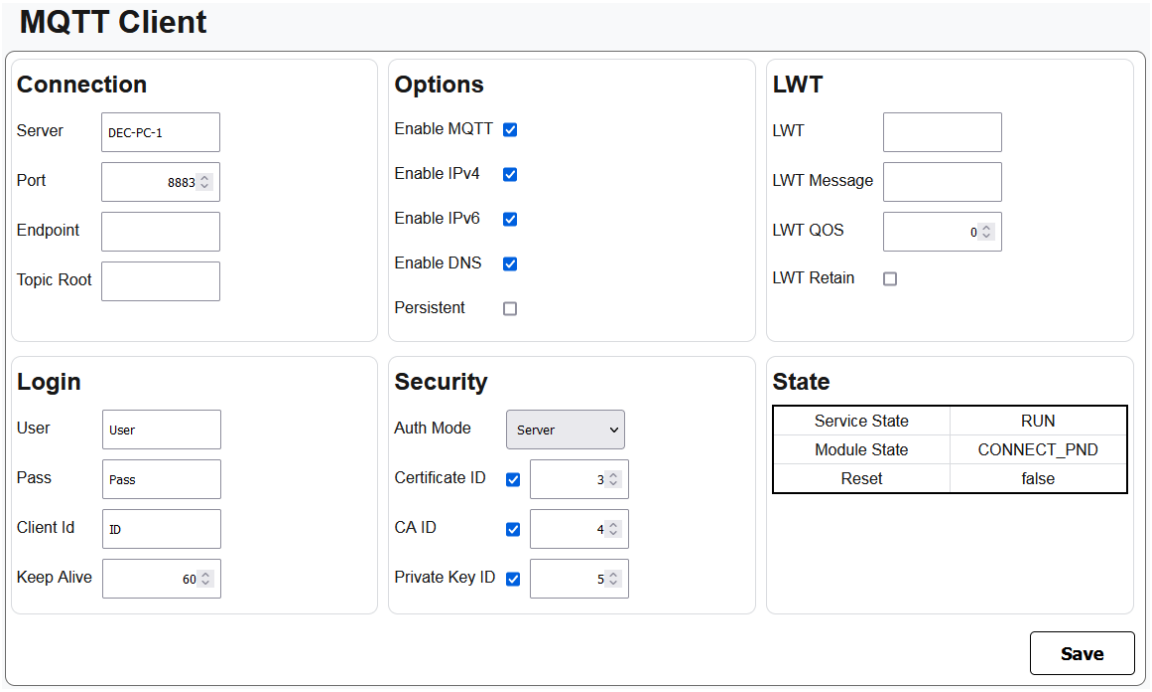



**Fig. 10 MQTT Client Configuration and State Page**

Only users with admin rights are allowed to access this page. It is not visible for other users.

Connection Configuration

| Parameter | |
|---|---|
| Server | IP address or the name of the MQTT server<br>If a server name is provided, *Enable DNS* must be activated. |
| Port | Port of the MQTT server<br>Default values are 1883 in case of unsecured server, 8883 in case of TLS secured server |
| Enable MQTT | Enable MQTT client |
| Enable IPv4 | Enable IPv4 addresses for MQTT client |
| Enable IPv6 | Enable IPv6 addresses for MQTT client |
| Enable DNS | Enable DNS for MQTT client.<br>Sever names are resolved by DNS.<br>DNS must also be activated in the network configuration (→ 3.3.5). |
| Persistent | Create persistent session<br>In case of connection loss, the server saves the persistent session and restores it when the client reconnects. |
| Topic Root | Topic root name<br>All topics are starting with the topic root name. Topics of a device can be grouped to a common namespace. |
| LWT Topic | Last will and testament topic |
| LWT Message | Last will and testament message string |
| LWT QoS | Last will and testament quality of service |
| LWT Retain | Last will and testament retain flag |

**Tab. 8 MQTT Client Connection Configuration**

Login Configuraton

| Parameter | |
|---|---|
| User | Login user name |
| Pass | Login password |
| Client Id | Login client identifier |
| Keep Alive | Keep Alive polling interval in seconds.<br>If no communication happened during keep alive time, the client polls the server by sending a PINGREQ. The keep alive interval can detect unreachable server situation.<br>Value 0 disables keep alive function. |

**Tab. 9 MQTT Client Login Configuration**

Security Configuration

<table>
<tr><th>Parameter</th><th></th></tr>
<tr><td>Auth Mode</td><td>Authentication mode during TLS.
<table>
<tr><td>Disable</td><td>Authentication disabled, unencrypted communication</td></tr>
<tr><td>Skip</td><td>Client does not verify server certificate.<br>Encrypted connection.</td></tr>
<tr><td>Server</td><td>Client verifies server certificate.<br>Encrypted connection.<br>Requires a valid CA or intermediate certificate.</td></tr>
<tr><td>Server + Client</td><td>Client verifies server certificate and server verifies client certificate. (2-way authentication).<br>Requires client certificate, CA or intermediate certificate and private key.</td></tr>
</table></td></tr>
<tr><td>Certificate ID</td><td>If checkbox is enabled, the value contains the identifier of certificate in secure memory (→ 3.4.6.1).<br>If the checkbox is disabled, no certificate is provided.</td></tr>
<tr><td>CA ID</td><td>If checkbox is enabled, the value contains the identifier of the CA or intermediate certificate in secure memory (→ 3.4.6.1).<br>If the checkbox is disabled, no certificate is provided.</td></tr>
<tr><td>Private Key ID</td><td>If checkbox is enabled, the value contains the identifier of the private key in secure memory (→ 3.4.6.1).<br>If the checkbox is disabled, no private key is provided.</td></tr>
</table>

**Tab. 10 MQTT Client Security Configuration**

Settings of *Certificate ID*, *CA ID* and *Private Key ID* are only relevant if *Auth Mode* is set to Server or Server + Client.

Authentication mode must be set up according to the requirements of the MQTT broker.

The configuration is updated by clicking the *Save* button. Updating parameters may require a restart of the device. In this case, the current configuration remains active and the Service State flag *Reset* is set to true. The new configuration becomes active after a restart.

Extended Configuration

Extended configuration with MQTT broker specific settings.

| Parameter | |
|---|---|
| Secure ID 0 | Configuration of a secure memory identifier. (→ 3.4.6.1)<br>Certificates and keys in Tab. 10 are used for TLS handshake only.<br>The parameter refers to a private key used for Google IoT JWT authentication. |
| Reconnect Interval | Reconnect time in minutes.<br>After this interval, LucidIoT disconnects from the server and reconnects immediately.<br>If 0, client does not disconnect and reconnect. |
| JWT Login | When active, the MQTT client calculates a JSON Web Token for authentication before it connects to the server. The password configured in the login configuration section is disregarded. |
| Login Data | MQTT broker specific login data |

**Tab. 11 MQTT Extended Configuration**

MQTT Client State

The section *State* provides information of the MQTT client state.

| Module State | |
|---|---|
| NONE | MQTT not enabled or configured |
| CON_DNS,<br>CON_DNS_PND | Connect with DNS (pending) |
| CONNECT,<br>CONNECT_PND | Connect to MQTT server (pending)<br>Wait until MQTT server responds |
| CONNECTED | Connected to MQTT server |
| DISCONNECT | Disconnecting from MQTT server |
| DISCONNECTED | Disconnected |
| ERROR | Error state |

**Tab. 12 MQTT Client Module States**

### 3.3.10.4 MQTT Topics

LucidIoT defines 16 MQTT topic items. Each topic item can be published periodically or be subscribed.


**Fig. 11 MQTT Topics Configuration Page**

Fig. 11 shows the configuration parameters for Topic item 0.

| Parameter | |
|---|---|
| Type | Type of the topic.<br>Disable: Topic is disabled<br>Publish: Topic is published periodically<br>Subscribe: Topic is subscribed |
| Name | Name of the topic<br>See also *Topic Root*. |
| PubInt | Publish interval in milliseconds (applies only if *Type* is Publish) |
| Value Id | Value identifier (→ 3.3.10.4.1) |
| QoS | Quality of service (0, 1, 2) |
| Format | Value format conversion (→ 3.3.10.4.4) |
| Retain | Retain flag is set when topic is published |

**Fig. 12 MQTT Topic Item Configuration**

Settings are updated by clicking the *Save Topics* button. Updates to MQTT topic settings are applied immediately and do not require a restart.

#### 3.3.10.4.1 Value Identifier

| Value Identifier | Address | |
|---|---|---|
| Ch 0 ... Ch 15 | 0 ... 15 | IO Channel Numbers (0 ... 15)<br>Input channels can be used for publish only<br>Output channels can be used for publish and subscribe |
| Ch 0 Param 0 ...<br>Ch 15 Param 0 | (1 << 5) +<br>0 ... 15 | Device function specific IO channel 0 ... 15 configuration parameter 0 |
| Ch 0 Param 1 ...<br>Ch 15 Param 1 | (2 << 5) +<br>0 ... 15 | Device function specific IO channel 0 ... 15 configuration parameter 1 |
| Ch 0 Param 2 ...<br>Ch 15 Param 2 | (3 << 5) +<br>0 ... 15 | Device function specific IO channel 0 ... 15 configuration parameter 2 |
| Ch 0 Param 3 ...<br>Ch 15 Param 3 | (4 << 5) +<br>0 ... 15 | Device function specific IO channel 0 ... 15 configuration parameter 3 |
| System Time | 0x1000 | System Time |
| System Date | 0x1001 | System Date |

**Tab. 13 Value Identifiers**

Tab. 13 lists all value identifiers (Value Ids). Value Ids link one LucidIoT value to a MQTT topic item or a data logging value.

Value Ids address the IO value of a single IO channel number, IO channel configuration parameter, or system values.

Example:
Assuming the digital output channel number 8 operates in duty-cycle mode.

The duty-cycle can be updated by the MQTT client accessing Param 1 which has the address (2 << 5) + 8 = 0x0048.

IO channel configuration parameters are only changed temporarily. They are not saved to the NV memory.

#### 3.3.10.4.2 Value Group Identifier (MQTT only)

| Value Identifier Group | Address | |
|---|---|---|
| Group<br>0 … 15 | 0x0100<br>…<br>0x010F | IO Channel Groups (0 … 15) |
| Status Message | 0x1F00 | Device status JSON formatted |

**Tab. 14 Value Group Identifiers**

An IO channel group gives access to multiple IO channel values by a single MQTT topic.

Each IO channel value is assigned to an IO channel group. An IO channel group can consist of multiple IO channel values.

By default, all IO channel values of IO slot 0 are assigned to value group identifier 0 and all IO channel values of IO slot 1 are assigned to value group identifier 1. All other value identifier groups are reserved.

Value group identifiers are only available for MQTT topics. They cannot be used for data logging.

In the case IO channels support multiple value types (e.g. RTD inputs which can return resistance or temperature values), the format specification is used in order to specify value type.

Values are formatted according to JSON standard (Tab. 154). Invalid values inside the JSON array are ignored and are reported to the system diagnostic.

The JSON is formatted according to the schema shown in Tab. 15.

| | |
|---|---|
| "Values":[<br>{"Id": valueId,<br>"Type": valueType,<br>"Value": value}, …, …<br>] | |

**Tab. 15 MQTT IO Value Group JSON Format**

The JSON shown in Tab. 16 is an example of the published IO Value Group 1 of a LIOT16-AI8-20MA-AI8-20MA with 16 0-20mA inputs.

The AI8-20MA input channels can return voltage (0-10V) or current (0-20mA) values. In the example currents are retuned in nA because the MQTT topic format was set to “Auto”.

Setting the format to some voltage configuration, returns all IO values in µV (Tab. 17).

<table>
<tr>
<td>

```
{
  "Values" : [ {
    "Id" : 8,
    "Type" : 35,
    "Value" : 1000
  }, {
    "Id" : 9,
    "Type" : 35,
    "Value" : 2000000
  }, {
    "Id" : 10,
    "Type" : 35,
    "Value" : 20000000
  }, {
    "Id" : 11,
    "Type" : 35,
    "Value" : 15000000
  }, {
    "Id" : 12,
    "Type" : 35,
    "Value" : 0
  }, {
    "Id" : 13,
    "Type" : 35,
    "Value" : 0
  }, {
    "Id" : 14,
    "Type" : 35,
    "Value" : 0
  }, {
    "Id" : 15,
    "Type" : 35,
    "Value" : 0
  } ]
}
```

</td>
<td>

```
Value Typ 35 -> 0x23 -> CUS4
1000nA = 0.001mA

2mA

20mA

15mA
```

</td>
</tr>
</table>

**Tab. 16 Value Group 1 Example (Current)**

<table>
<tr>
<td>

```
{
  "Values" : [ {
    "Id" : 8,
    "Type" : 29,
    "Value" : 1000
  },

…

  {
    "Id" : 15,
    "Type" : 29,
    "Value" : 10000000
  } ]
}
```

</td>
<td>

```
Value Typ 29 -> 0x1D -> VOS4
1000µV = 0.001V

10000000µV = 10V
```

</td>
</tr>
</table>

**Tab. 17 Value Group 1 Example (Voltage)**

#### 3.3.10.4.3 Status Message Topic

...

#### 3.3.10.4.4 Value Format

The value format defines how data is converted or interpreted (e.g. true/false for digital input or output IO channel values).

Value formats are used by MQTT when converting between payload (strings) and intern values (e.g. IO channel value).

Value formats are used by data logging service when converting an intern value to a string representation in the log file.

All values have an automatic value format. Configuration other than automatic value format is optional.

| Value Format | IO Channel Type | Conversion |
|---|---|---|
| Auto | Any | → Automatic Value Format |
| Digital On / Off | Digital Input<br>Digital Output | 0 to "Off", 1 to "On" |
| Digital 1 / 0 | | 0 to "0", 1 to "1" |
| Digital true / false | | 0 to "false, 1 to "true" |
| Numeric | Digital Input (Counter) | Counter value as string |
| Voltage (µV) Integer | Analog Voltage Input<br>Analog Voltage Output | Integer in µV as string |
| Voltage (V) Decimal | | Float in V as string |
| Voltage (V) Decimal Prec 0 | | Float in V with 0 decimal places as string |
| Voltage (V) Decimal Prec 1 | | Float in V with 1 decimal place as string |
| Voltage (V) Decimal Prec 2 | | Float in V with 2 decimal places as string |
| Voltage (V) Decimal Prec 3 | | Float in V with 3 decimal places as string |
| Current (nA) Integer | Analog Current Input<br>Analog Current Output | Integer in nA as string |
| Current (mA) Decimal | | Float in mA as string |
| Current (mA) Decimal Prec 0 | | Float in mA with 0 decimal places as string |
| Current (mA) Decimal Prec 1 | | Float in mA with 1 decimal place as string |
| Current (mA) Decimal Prec 2 | | Float in mA with 2 decimal places as string |
| Current (mA) Decimal Prec 3 | | Float in mA with 3 decimal places as string |
| Resistance (mΩ) Integer | RTD Input (Resistance) | Integer in mΩ as string |
| Resistance (Ω) Decimal | | Float in Ω as string |
| Resistance (Ω) Decimal Prec 0 | | Float in Ω with 0 decimal places as string |
| Resistance (Ω) Decimal Prec 1 | | Float in Ω with 1 decimal place as string |
| Resistance (Ω) Decimal Prec 2 | | Float in Ω with 2 decimal places as string |
| Resistance (Ω) Decimal Prec 3 | | Float in Ω with 3 decimal places as string |
| Temperature (m°C) Integer | RTD Input (Temperature) | Integer in m°C as string |
| Temperature (°C) Decimal | | Float in °C as string |
| Temperature (°C) Decimal Prec 0 | | Float in °C with 0 decimal places as string |
| Temperature (°C) Decimal Prec 1 | | Float in °C with 1 decimal place as string |

| | | |
|---|---|---|
| Temperature (°C) Decimal Prec 2 | | Float in °C with 2 decimal places as string |
| Temperature (°C) Decimal Prec 3 | | Float in °C with 3 decimal places as string |

**Tab. 18 List of Value Formats**

Note
The number of decimal places is only relevant for published MQTT topics and data logging.
Subscribed MQTT topics are accepted with any number of decimal places.
The character '.' (Point) is used as decimal separator.

Automatic Value Format
By default, the automatic value format is used. It detects the value format by the type of the IO channel.

| IO Channel Type | Automatic Value Format |
|---|---|
| Digital Input<br>Digital Output | Digital 1 / 0 |
| Analog Voltage Input<br>Analog Voltage Output | Voltage (V) decimal |
| Analog Current Input | The AI modules measure voltages and currents.<br>Automatic value format is voltage (V) decimal |
| Analog Current Output | Current (mA) decimal |
| RTD Input | Temperature (°C) |

**Tab. 19 Automatic Value Format**

Invalid Value Id and Value Format configurations
If an invalid combination of value identifier and value format is configured, the conversion fails and data is not published or logged. Subscribed data is detected as malformed and is ignored.

In this case, the topic statistic (→3.3.10.4.5) and the system log (→3.4.9) should be consulted for further details.

### 3.3.10.4.5 MQTT Topic Statistic

The MQTT Topic Statistic shows the internal state of each topic.

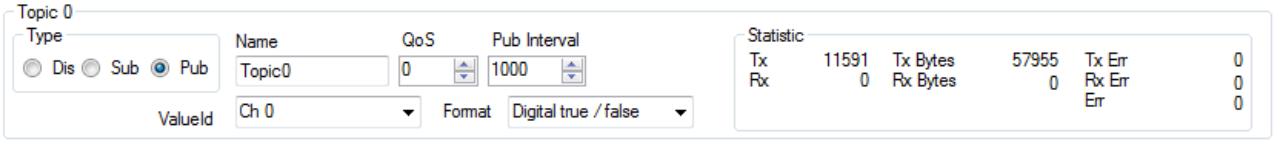



**Fig. 13 MQTT Topic Statistic**

Fig. 13 shows the configuration and statistic of a published MQTT Topic. It shows the number of received and transmitted data packets and bytes. It shows also the number transmission and value conversion errors.

The System Log function (→3.4.9) logs and resets the MQTT topic statistic daily.

#### 3.3.10.5 MQTT Broker Configuration

This section explains the configuration of popular MQTT brokers.

MQTT brokers must be set up according to their instructions.

Prerequisites and Assumptions:

- Device is set up in the MQTT broker.
  - Best practice is to test the MQTT broker connection with MQTT clients on the PC in advance.
- Private key and certificate have been created and are stored in LucidIoT Secure Memory.

MQTT brokers may only support a subset of the MQTT specification. Please consult the documentation of the services for further information.

##### 3.3.10.5.1 Eclipse Mosquitto

Eclipse Mosquitto is compatible with MQTT Standard 3.1.1.

##### 3.3.10.5.2 Amazon AWS

…

##### 3.3.10.5.3 Google Cloud Platform (IoT Core)

A connection to GCP IoT Core is TLS secured. The client can validate the server certificate by configuring the CA. Google IoT does not require the client sending its certificate.

GCP IoT Core uses JSON Web Token (JWT) authentication. LucidIoT calculates the JWT before it connects to the service. This requires an accurate system time. Otherwise, GCP IoT Core would deny login. It is recommended to synchronize the system clock.

GCP IoT Core disconnects the device after 24h. LucidIoT can reconnect after that interval automatically. When disconnecting it is possible that data is lost during.

Connections to GCP need configuration of the CA certificate which can be downloaded. The TLS should check the validity of the server certificate.

The JWT is signed by a private key which is configured by the Secure Id 0 parameter.

| Parameter | Notes |
|---|---|
| Server | mqtt.googleapis.com |
| Port | 8883 |
| Auth Mode | Authentication Mode.<br>When set to “Server” the client checks server certificate (recommended). CA must be configured<br>When set to “Skip” the client does not check server certificate. Configuration of CA not needed.<br>In both modes the connection is encrypted.<br>Other authentication modes are not supported by GCP. |
| CA Id | Secure Memory Id of CA certificate.<br>Needs to be set up for authentication mode “Server”. |
| Certificate Id | - |
| Private Key Id | - |
| User | Can be empty or set to “ignore” |
| Pass | Setting is ignored. When JWT Login is enabled the JWT overwrites this setting. |
| Client Id | Client Id in the format:<br>projects/*PROJECT_ID*/locations/*REGION*/registries/*REGISTRY_ID*/devices/*DEVICE_ID* |
| Broker Data | Broker specific data<br>Set to PROJECT_ID |
| Secure Id 0 | Secure Memory Identifier for the RSA key used for signing JWT |
| ReCon Interval | Reconnect interval.<br>GCP disconnects after 1440 min (24h). A shorter interval can be configured. JWT considers reconnect interval and calculates expiry timestamp accordingly. |
| JWT Login | Must be enabled for GCP connections |

**Tab. 20 GCP IoT Core Configuration Parameters**

| Type of Topic | Topic Name |
|---|---|
| Publish | /devices/*DEVICE_ID*/events/*TopicName* |
| Subscribe | /devices/*DEVICE_ID*/commands/# |

**Tab. 21 GCP IoT Core Topics**

Tab. 21 lists the topic names for published and subscribed topics supported by GCP. GCP does not support arbitrary topic names. Topics need to be set up in GCP Pub/Sub.

While a client can publish to user created topics it is only possible for a client to subscribe limited topics.

This means that only by the topic commands/# can be subscribed by the MQTT client. The value identifier groups (→ 3.3.10.4.2) can receive JSON data with an array of values.

#### 3.3.10.5.4 Microsoft Azure IoT Hub

| Parameter | Notes |
|---|---|
| Server | Server hostname<br>*hubname*.azure-devices.net |
| Port | 8883 |
| Auth Mode | Authentication Mode.<br>When set to "Server" the client checks server certificate (recommended). CA must be configured<br>When set to "Skip" the client does not check server certificate. Configuration of CA not needed.<br>In both modes the connection is encrypted.<br>Other authentication modes are not supported by GCP.<br>When set to "Server + Client" server and client are authenticated. |
| CA Id | Secure Memory Id of CA certificate.<br>CA which issued the azure-devices.net server certificate.<br>E.g. CN= Baltimore CyberTrust Root |
| Certificate Id | Secure Memory Id of the device certificate |
| Private Key Id | Secure Memory Id of the device public kdey |
| User | Login User Name<br>*hubname*.azure-devices.net/*device_id*/?api-version=2018-06-30 |
| Pass | Login Password<br>Empty |
| Client Id | Client Id<br>Set to *device_id* |
| Broker Data | - |
| Secure Id 0 | - |
| ReCon Interval | Reconnect interval.<br>Can be set to 0 or any desired interval. |
| JWT Login | Disabled |

**Tab. 22 Azure IoT Hub Configuration Parameters**

| Type of Topic | Topic Name |
|---|---|
| Publish | devices/*device_id*/messages/events/ |
| Subscribe | devices/*device_id*/messages/devicebound/# |

**Tab. 23 Azure IoT Hob Topics**

Does not support Qos 2 and retains.

## 3.3.11 Modbus/TCP Server

The Modbus/TCP field bus protocol is based on top of the Modbus application layer. It uses TCP/IP protocol and Ethernet link layer instead of the serial transfer used by Modbus RTU or ASCII.

LucidIoT can operate as Modbus/TCP server, which allows clients to access to IO channel values, some IO channel configuration parameters and system values.

The Modbus/TCP server has full access to LucidIoT value identifiers. The user account management (→ 3.4.5) is disabled for this service.

Modbus/TCP server is disabled by default.

### 3.3.11.1 Configuration

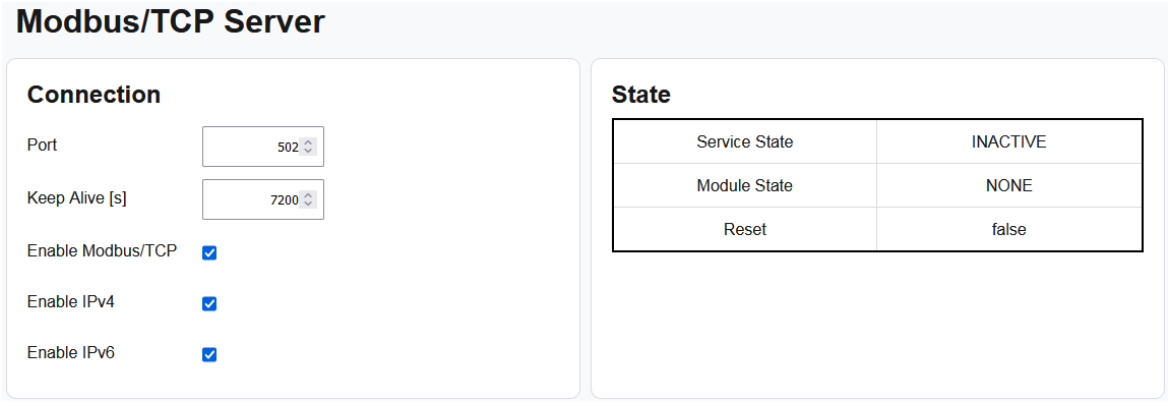



**Fig. 14 Modbus/TCP Server Configuration and State Page**

Only users with admin rights are allowed to access this page. It is not visible for other users.

Connection Configuration

| Parameter | |
|---|---|
| Port | Port of the Modbus/TCP server<br>Default values is 502 |
| Keep Alive | Keep Alive Time in seconds (→ 3.3.7.1) |
| Enable Modbus/TCP | Enable Modbus/TCP server |
| Enable IPv4 | Enable IPv4 addresses for Modbus/TCP server |
| Enable IPv6 | Enable IPv6 addresses for Modbus/TCP server |

**Tab. 24 Modbus/TCP Server Connection Configuration**

The configuration is updated by clicking the *Save* button. Updating parameters may require a restart of the device. In this case, the current configuration remains active and the Service State flag *Reset* is set to true. The new configuration is updated after a restart.

The section *State* provides information of the Modbus/TCP server service state.

#### 3.3.11.2 Modbus Registers

| Address | Type | Value width | Description |
|---|---|---|---|
| 0x1000 | Holding | 32 | IO Channel Number 0 Value |
| 0x1002 | Holding | 32 | IO Channel Number 1 Value |
| 0x1004 | Holding | 32 | IO Channel Number 2 Value |
| 0x1006 | Holding | 32 | IO Channel Number 3 Value |
| 0x1008 | Holding | 32 | IO Channel Number 4 Value |
| 0x100A | Holding | 32 | IO Channel Number 5 Value |
| 0x100C | Holding | 32 | IO Channel Number 6 Value |
| 0x100E | Holding | 32 | IO Channel Number 7 Value |
| 0x1010 | Holding | 32 | IO Channel Number 8 Value |
| 0x1012 | Holding | 32 | IO Channel 9 Number Value |
| 0x1014 | Holding | 32 | IO Channel Number 10 Value |
| 0x1016 | Holding | 32 | IO Channel Number 11 Value |
| 0x1018 | Holding | 32 | IO Channel Number 12 Value |
| 0x101A | Holding | 32 | IO Channel Number 13 Value |
| 0x101C | Holding | 32 | IO Channel Number 14 Value |
| 0x101E | Holding | 32 | IO Channel Number 15 Value |
| 0x1080 ... 0x109E | Holding | 32 | Alternative IO Channel Number 0 ... 15 Value |
| 0x1100 | Holding | 32 | IO Channel Number 0 Configuration Param 0 |
| 0x1102 | Holding | 32 | IO Channel Number 1 Configuration Param 0 |
| 0x1104 | Holding | 32 | IO Channel Number 2 Configuration Param 0 |
| 0x1106 | Holding | 32 | IO Channel Number 3 Configuration Param 0 |
| 0x1108 | Holding | 32 | IO Channel Number 4 Configuration Param 0 |
| 0x110A | Holding | 32 | IO Channel Number 5 Configuration Param 0 |
| 0x110C | Holding | 32 | IO Channel Number 6 Configuration Param 0 |
| 0x110E | Holding | 32 | IO Channel Number 7 Configuration Param 0 |
| 0x1110 | Holding | 32 | IO Channel Number 8 Configuration Param 0 |
| 0x1112 | Holding | 32 | IO Channel Number 9 Configuration Param 0 |
| 0x1114 | Holding | 32 | IO Channel Number 10 Configuration Param 0 |
| 0x1116 | Holding | 32 | IO Channel Number 11 Configuration Param 0 |
| 0x1118 | Holding | 32 | IO Channel Number 12 Configuration Param 0 |
| 0x111A | Holding | 32 | IO Channel Number 13 Configuration Param 0 |
| 0x111C | Holding | 32 | IO Channel Number 14 Configuration Param 0 |
| 0x111E | Holding | 32 | IO Channel Number 15 Configuration Param 0 |
| 0x1200 | Holding | 32 | IO Channel Number 0 Configuration Param 1 |
| 0x1202 | Holding | 32 | IO Channel Number 1 Configuration Param 1 |

| | | | |
|---|---|---|---|
| 0x1204 | Holding | 32 | IO Channel Number 2 Configuration Param 1 |
| 0x1206 | Holding | 32 | IO Channel Number 3 Configuration Param 1 |
| 0x1208 | Holding | 32 | IO Channel Number 4 Configuration Param 1 |
| 0x120A | Holding | 32 | IO Channel Number 5 Configuration Param 1 |
| 0x120C | Holding | 32 | IO Channel Number 6 Configuration Param 1 |
| 0x120E | Holding | 32 | IO Channel Number 7 Configuration Param 1 |
| 0x1210 | Holding | 32 | IO Channel Number 8 Configuration Param 1 |
| 0x1212 | Holding | 32 | IO Channel Number 9 Configuration Param 1 |
| 0x1214 | Holding | 32 | IO Channel Number 10 Configuration Param 1 |
| 0x1216 | Holding | 32 | IO Channel Number 11 Configuration Param 1 |
| 0x1218 | Holding | 32 | IO Channel Number 12 Configuration Param 1 |
| 0x121A | Holding | 32 | IO Channel Number 13 Configuration Param 1 |
| 0x121C | Holding | 32 | IO Channel Number 14 Configuration Param 1 |
| 0x121E | Holding | 32 | IO Channel Number 15 Configuration Param 1 |
| 0x2000 ... 0x200F | Holding | 16 | IO Channel Number 0 ... 15 Value |
| 0x2080 ... 0x208F | Holding | 16 | Alternative IO Channel Number 0 ... 15 Value |
| 0x2100 ... 0x210F | Holding | 16 | IO Channel Number 0 ... 15 Configuration Param 0 |
| 0x2200 ... 0x220F | Holding | 16 | IO Channel Number 0 ... 15 Configuration Param 1 |
| 0x8000 | Input | 16 | System Date Year |
| 0x8001 | Input | 16 | System Date Month |
| 0x8002 | Input | 16 | System Date Day |
| 0x8003 | Input | 16 | System Date Hour |
| 0x8004 | Input | 16 | System Date Minute |
| 0x8005 | Input | 16 | System Date Second |

**Tab. 25 Modbus/TCP Register**

Register value format

Modbus/TCP input and holding registers are either 32 or 16 bit wide.
32 bit registers consist of two consecutive 16 bit registers representing a value in big-endian format.

For all 32 bit registers, a compatible 16 bit register is also accessible. 16 bit registers may have some constraints, e.g. lower resolution.

32 bit registers start at address 0x1000, 16 bit registers at 0x2000.

Example
The value of the analog input (or output) channel 0 is 1.234567 V = 1234567 µV (1234 mV).
The hexadecimal representation of the value is 0x0012D687 (1234567) and 0x04D2 (1234).
Register 0x1000 contains 0x0012, Register 0x1001 contains 0xD687
Register 0x2000 contains 0x04D2 (16 bit registers contain voltages in mV)

### 3.3.12 FTP Server

The FTP server accesses the data stored on the file system (3.4.2).

FTP logins are controlled by the user account management (→ 3.4.5). Users need FTP access rights in order to log in.

The FTP server supports active and passive mode.

File transfers must operate in binary format.

Because of limited resources, FTP clients must open one connection to the FTP server (e.g. when transmitting files in a queue).

The FTP server supports implicit security mode. If implicit security is enabled, TLS handshake is performed for all connections and communication is encrypted. Implicit security needs set up of certificates and keys.

Explicit security mode is not supported.

FTP is disabled by default.

### 3.3.12.1 FTP Server Configuration

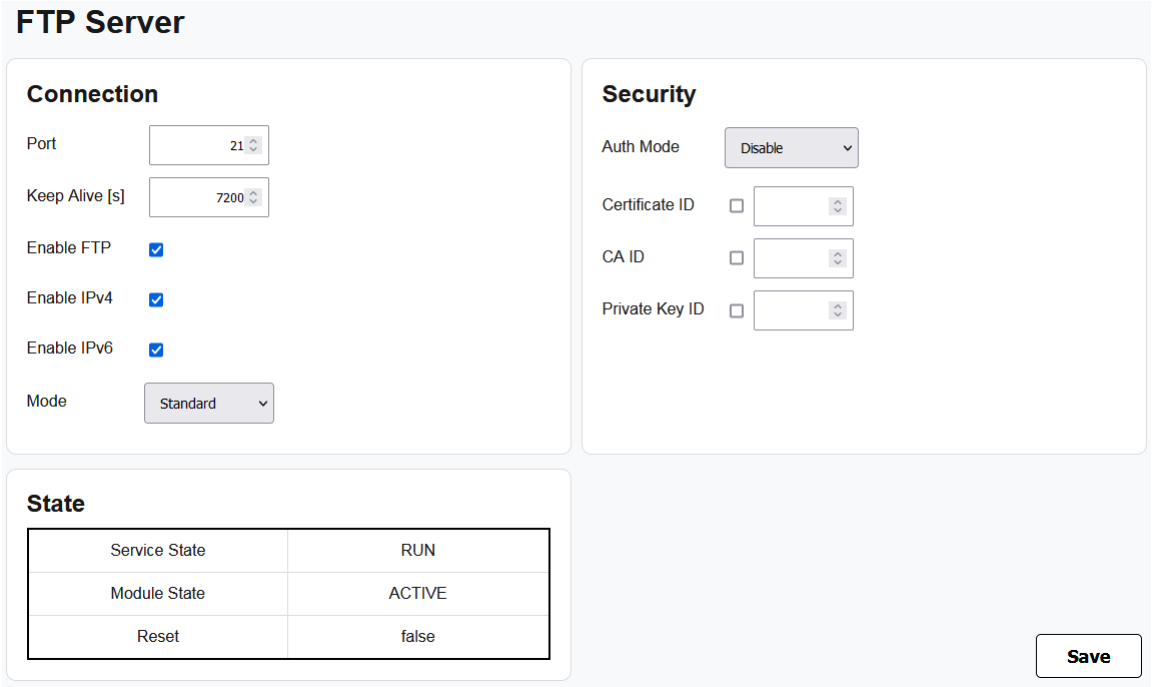



**Fig. 15 FTP Server Configuration and State Page**

Only users with admin rights are allowed to access this page. It is not visible for other users.

Connection Configuration

| Parameter | |
|---|---|
| Port | Port of the FTP server<br>Default values are 21 in standard mode, 990 in implicit mode. |
| Keep Alive | Keep Alive Time in seconds (→ 3.3.7.1) |
| Enable FTP | Enable FTP server |
| Enable IPv4 | Enable IPv4 addresses for FTP server |
| Enable IPv6 | Enable IPv6 addresses for FTP server |
| Mode | FTP mode.<br>Standard mode without TLS<br>Implicit mode with TLS. |

**Tab. 26 FTP Server Configuration**

Security Configuration

| Parameter | |
|---|---|
| Certificate ID | If the checkbox is enabled, the value contains the identifier of certificate in secure memory (→ 3.4.6.1).<br>If the checkbox is disabled, no certificate is provided. |
| CA ID | If the checkbox is enabled, the values contains the identifier of the CA or intermediate certificate in secure memory (→ 3.4.6.1).<br>If the checkbox is disabled, no certificate is provided. |

| | |
|---|---|
| Private Key ID | If the checkbox is enabled, the value contains the identifier of the private key in secure memory (→ 3.4.6.1)<br>If the checkbox is disabled, no private key is provided. |

**Tab. 27 FTP Server Security Configuration**

Settings of *Certificate ID*, *CA ID*, *Private Key ID* are only relevant if *Mode* is set to *Implicit* and *Auth Mode* is set to *Server*.

The configuration is updated by clicking the *Save* button. Updating parameters may require a restart of the device. In this case, the current configuration remains active and the *Service State* flag *Reset* is set to true. The new configuration is updated after a restart.

The section *State* provides information of the service state.

### 3.3.12.2 FTP Client Configuration

This section shows the configuration of FileZilla FTP client.

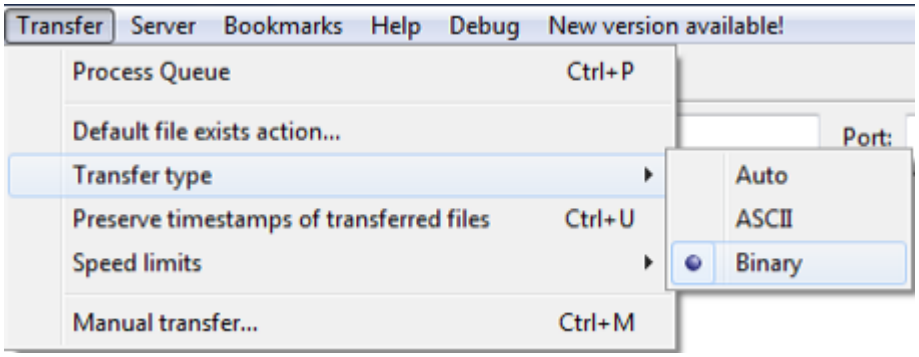



**Fig. 16 FileZilla Transfer Type**

The FTP server supports binary file transmission only. The configuration of transfer type in Fig. 16 instructs the FTP client using binary transfer mode for all type of files.

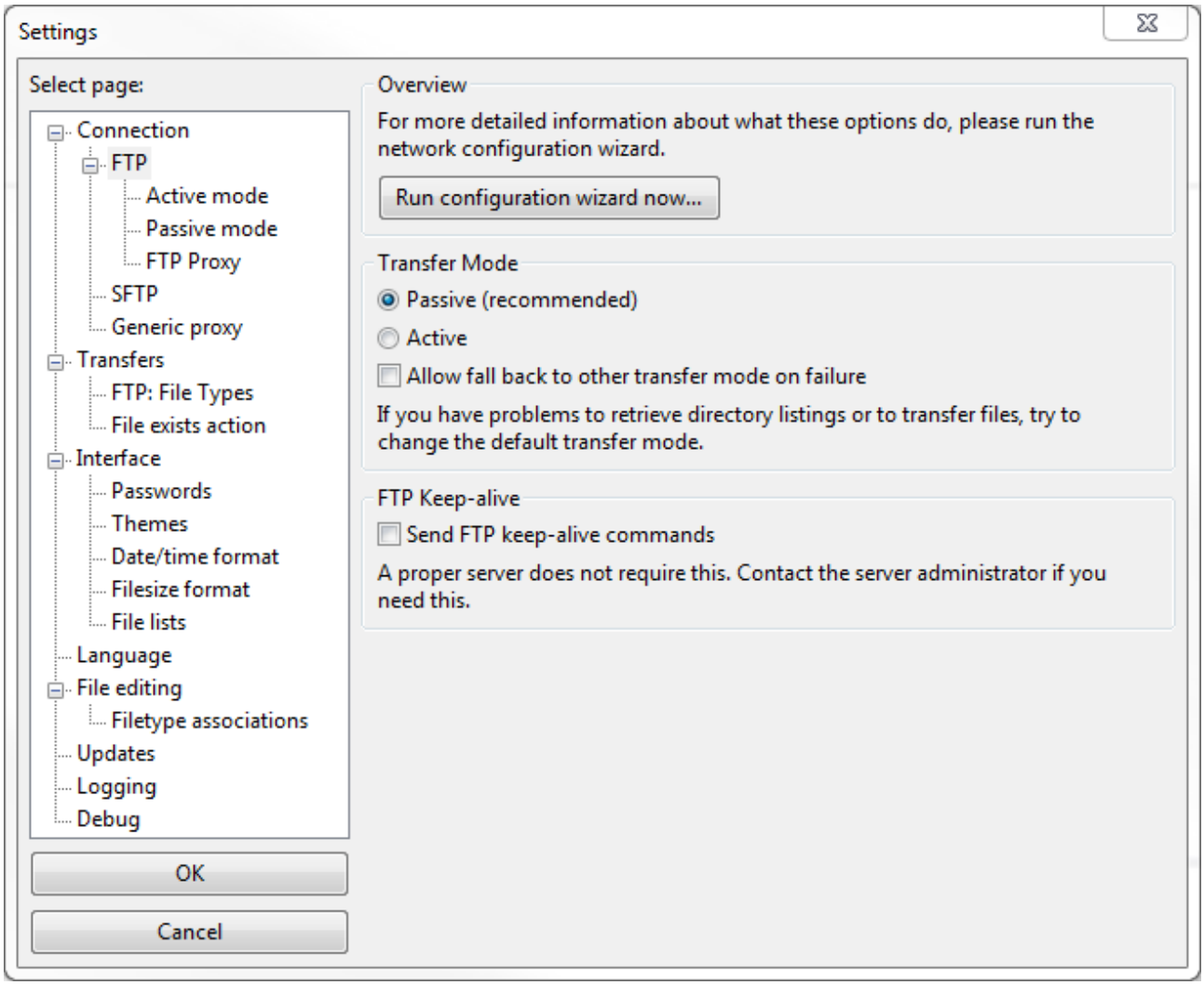



**Fig. 17 FileZilla Transfer Mode**

The FTP server supports passive and active transfer mode.
In passive transfer mode, the FTP server listens for an incoming data connection.
In active mode, the FTP server initiates an outgoing data connection.

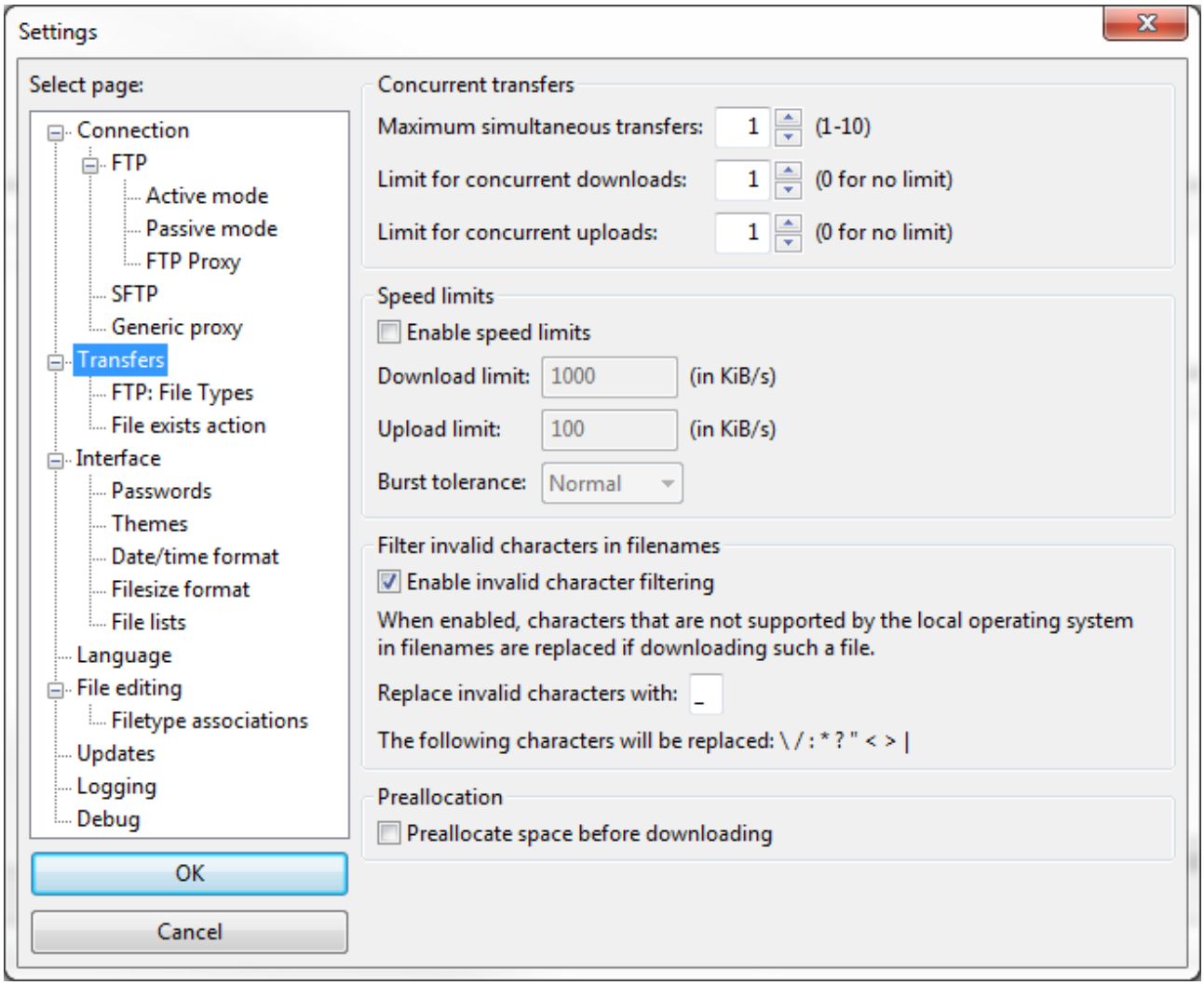



**Fig. 18 FileZilla Transfer Limits**

Because of resource constraints, the FTP client should be configured using only one connection in order to prevent parallel transfers.

We recommend limiting the number of connections to 1 (→ Fig. 18).

A connection to the FTP server is established with FileZilla by:
admin@ftp://192.168.177.240

A connection with implicit security mode is established with FileZilla by:
admin@ftps://192.168.177.240

### 3.3.13 SNTP

The system clock (→ 3.4.4) can be synchronized by SNTP.

If *Enable SNTP* and *Sync Clock* are enabled, the system clock is under control of SNTP and is synchronized every hour.

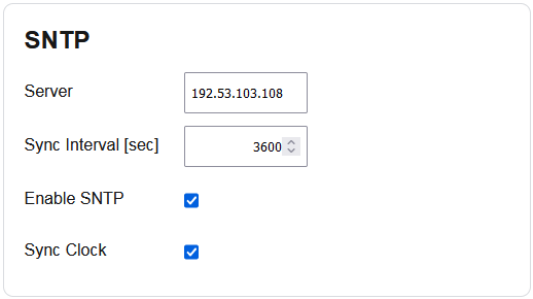



**Fig. 19 SNTP Configuration**

Fig. 19 shows the section with configuration settings of SNTP on the system clock configuration page (→ Fig. 26).

SNTP Configuration

| Parameter | |
|---|---|
| Server | SNTP server name<br>IPv4 address of the SNTP timeserver |
| Sync Interval | Synchronization Interval in seconds<br>This setting is currently disregarded. The system clock will be synchronized every 3600 s |
| Enable SNTP | If set, SNTP is enabled |
| Sync Clock | If set and SNTP enabled, system clock is synchronized |

**Tab. 28 SNTP Configuration**

If the system clock is synchronized with a timeserver, the value *Sync* in the State section (→ Fig. 26) is set to true. If the system clock is not synchronized with a timeserver of the connection to the timeserver got lost, *Sync* is set to false.

### 3.3.14 JSON Interface

The JSON interface server is listening on a configurable TCP port. Clients can connect to the server and send requests.

The JSON interface uses the JSON protocol (→ 5.3). This is the recommended way to exchange data with LucidIoT. The provided API uses the JSON interface.

The JSON interface gives full access to all functions of LucidIoT and is under control of the user account management (→3.4.5).

The JSON interface is always enabled.

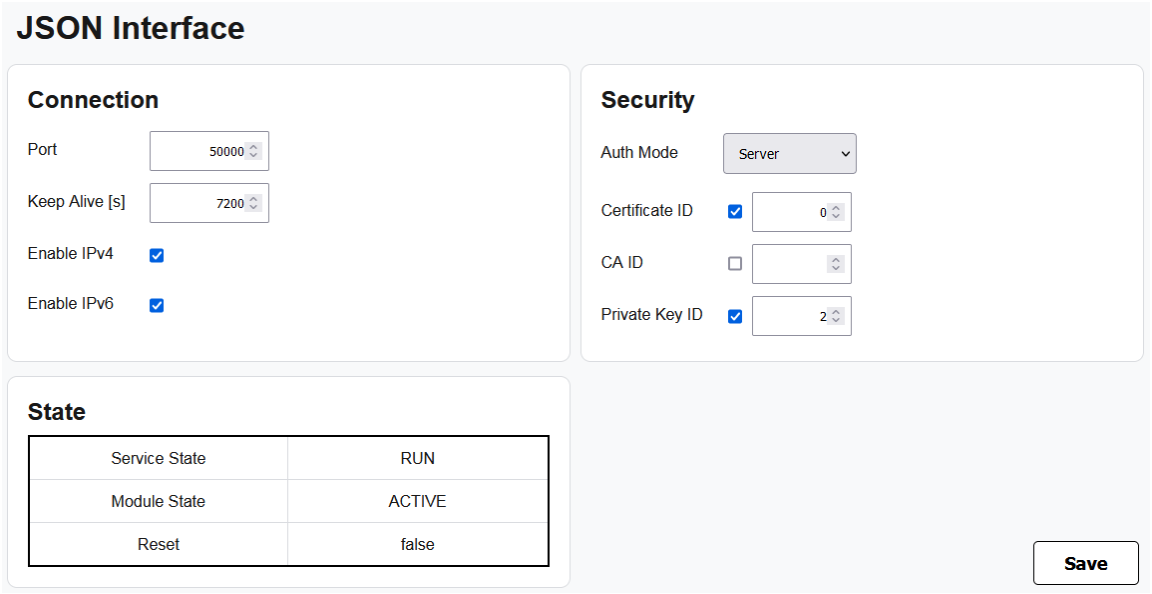



**Fig. 20 JSON Interface Configuration and State Page**

Only users with admin rights are allowed to access this page. It is not visible for other users.

JSON Interface Configuration

| Parameter | |
|---|---|
| Port | JSON interface port |
| Keep Alive | Keep Alive Time in seconds (→ 3.3.7.1) |
| Enable IPv4 | Enable IPv4 addresses for JSON interface |
| Enable IPv6 | Enable IPv6 addresses for JSON interface |

**Tab. 29 JSON Interface Configuration Parameters**

Security Configuration

| Parameter | |
|---|---|
| Auth Mode | JSON interface server authentication mode<br>If set to Disable, connection is not secured.<br>If set to Server, connection is TLS secured and the JSON interface server sends its certificates to the host.<br>The certificate of the host is not checked by the server in any case. |
| Certificate ID | If checkbox is enabled, the value contains the identifier of certificate in secure memory (→ 3.4.6.1).<br>If the checkbox is disabled, no certificate is provided.<br>Effective only if Auth Mode set to Server. |
| CA ID | If the checkbox is enabled, the value contains the identifier of the CA or intermediate certificate in secure memory (→ 3.4.6.1).<br>If the checkbox is disabled, no certificate is provided.<br>Effective only if Auth Mode set to Server. |

| | |
|---|---|
| Private Key ID | If the checkbox is enabled, the value contains the identifier of the private key in secure memory (→ 3.4.6.1).<br>If the checkbox is disabled, no private key is provided.<br>Effective only if Auth Mode set to Server. |
| Require IO Login | If this checkbox is enabled, IO read/write functions require login and are under control of the user account management (→3.4.5) |
| Require IO Config | If this checkbox is enabled, IO configuration parameter access requires login and are under control of the user account management (→3.4.5) |

**Tab. 30 JSON Interface Security Configuration Parameters**

The configuration is updated by clicking the *Save* button. Updating parameters may require a restart of the device. In this case, the current configuration remains active and the Service State flag *Reset* is set to true. The new configuration is updated after a restart.

### 3.3.15 FRAME Interface

The FRAME interface uses the compatible byte-oriented FRAME Protocol (→ 5.4) which is used by other products of the LucidControl family. The LucidIoCtrl command line tool can be used in order to access LucidIoT.

The FRAME interface gives limited access to IO config and IO value command group (→ 5.1).

Accessing IO channel values or IO channel configurations is not controlled by the user account management (→3.4.5). The FRAME interface has full access rights.

The FRAME interface is disabled by default.

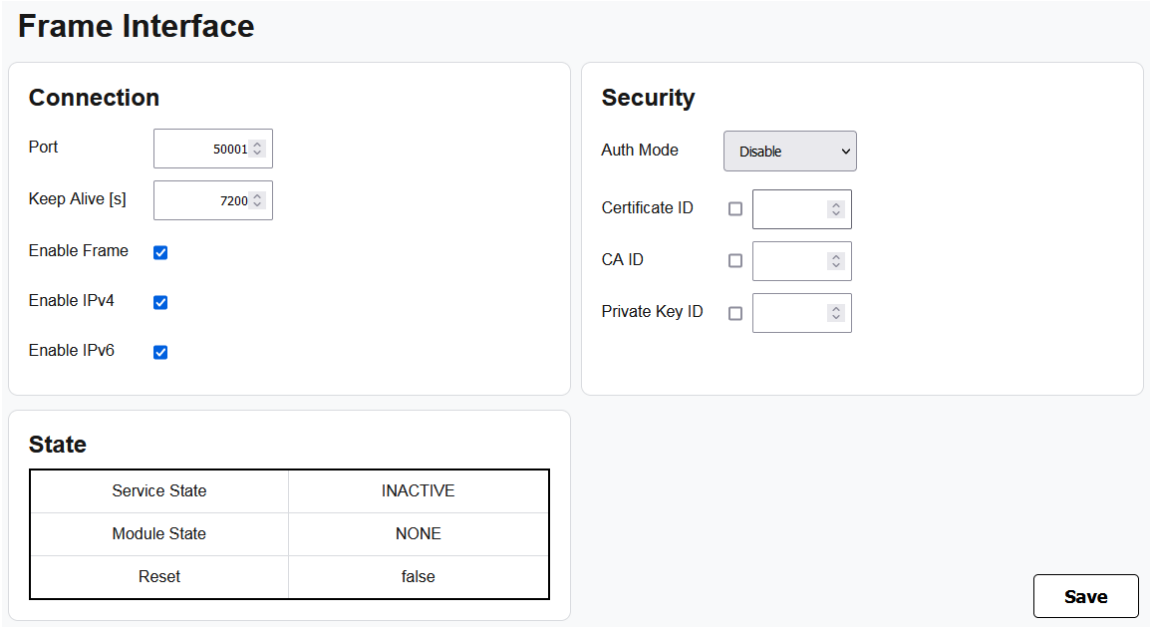



**Fig. 21 FRAME Interface Configuration and State Page**

Only users with admin rights are allowed to access this page. It is not visible for other users.

FRAME Interface Configuration

| Parameter | |
|---|---|
| Port | FRAME interface port |
| Keep Alive | Keep Alive Time in seconds (→ 3.3.7.1) |
| Enable IPv4 | Enable IPv4 addresses for FRAME interface |
| Enable IPv6 | Enable IPv6 addresses for FRAME interface |

**Tab. 31 FRAME Interface Configuration Parameters**

Security Configuration

| Parameter | |
|---|---|
| Aut Mode | Server authentication mode<br>If set to Disable, connection is not secured.<br>If set to Server, connection is TLS secured. |
| Certificate ID | If the checkbox is enabled, the value contains the identifier of certificate in secure memory (→ 3.4.6.1).<br>If the checkbox is disabled, no certificate is provided. |
| CA ID | If the checkbox is enabled, the value contains the identifier of the CA or intermediate certificate in secure memory (→ 3.4.6.1).<br>If the checkbox is disabled, no certificate is provided. |
| Private Key ID | If the checkbox is enabled, the value contains the identifier of the private key in secure memory (→ 3.4.6.1).<br>If the checkbox is disabled, no private key is provided. |

**Tab. 32 FRAME Interface Security Configuration Parameters**

Settings of *Certificate ID*, *CA ID*, *Private Key ID* are only relevant if *Auth Mode* is set to Server.

The configuration is updated by clicking the *Save* button. Updating parameters may require a restart of the device. In this case, the current configuration remains active and the *Service State* flag *Reset* is set to true. The new configuration is updated after a restart.

### 3.3.16 Limitations

LucidIoT as a low power embedded system offers limited resources. Compared to personal computers or e.g. the Raspberry Pi, LucidIoT offers far less resources regarding memory or processing speed.

LucidIoT can handle only a limited number of simultaneous incoming and outgoing network connections. The number or parallel TCP connections is limited to 30.

In the case that many network connections were established, this limit may be reached. Connections with activated TLS must also be considered as they consume additional memory.

This example use case works without limitation:

- 1 TLS secured connection outgoing to MQTT server
- 1 not secured connections for FTP file transfer
- 1 not secured connections for Modbus server
- 4 TLS secured connections to Web browsers (2 browsers)
- 2 TLS secured connections to the JSON interface
- 2 not secured connections to FRAME interface
- SNTP connection

LucidIoT is able to handle more connections than the listed ones, but the list should indicate roughly, what performance can be expected.

If you intend using LucidIoT in an application, which differs much from the explained use case, please contact us.

The system status overview (→ 3.4.1) shows important information about the available and occupied resources.

Unused connections should be closed by the client in order to free the resources on LucidIoT.

By using keep alive messages (→ 3.3.7.1), LucidIoT can check for the accessibility of a connected peer. In case of unreachable peers, LucidIoT may free the resources.

## 3.4 General Services

### 3.4.1 System Information

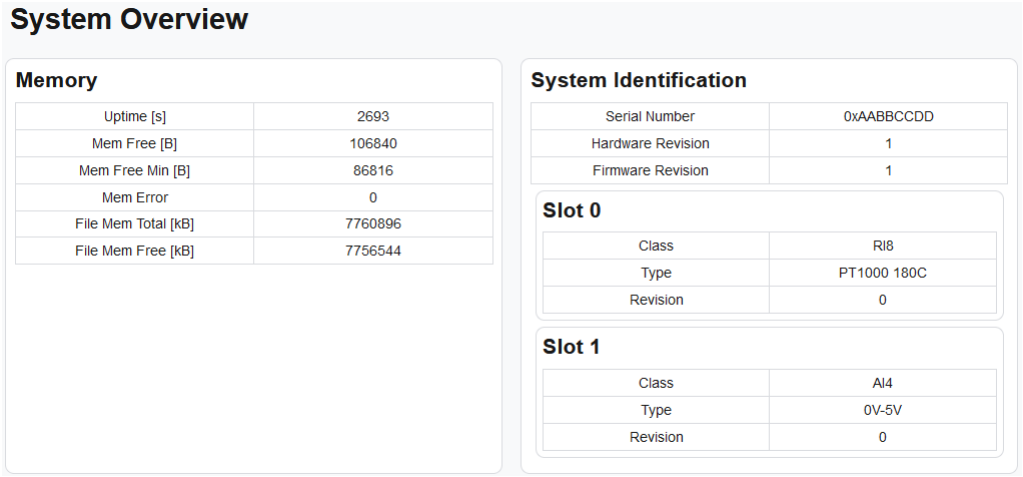



**Fig. 22 System Status Information**

The system information page shows device status details.

Memory

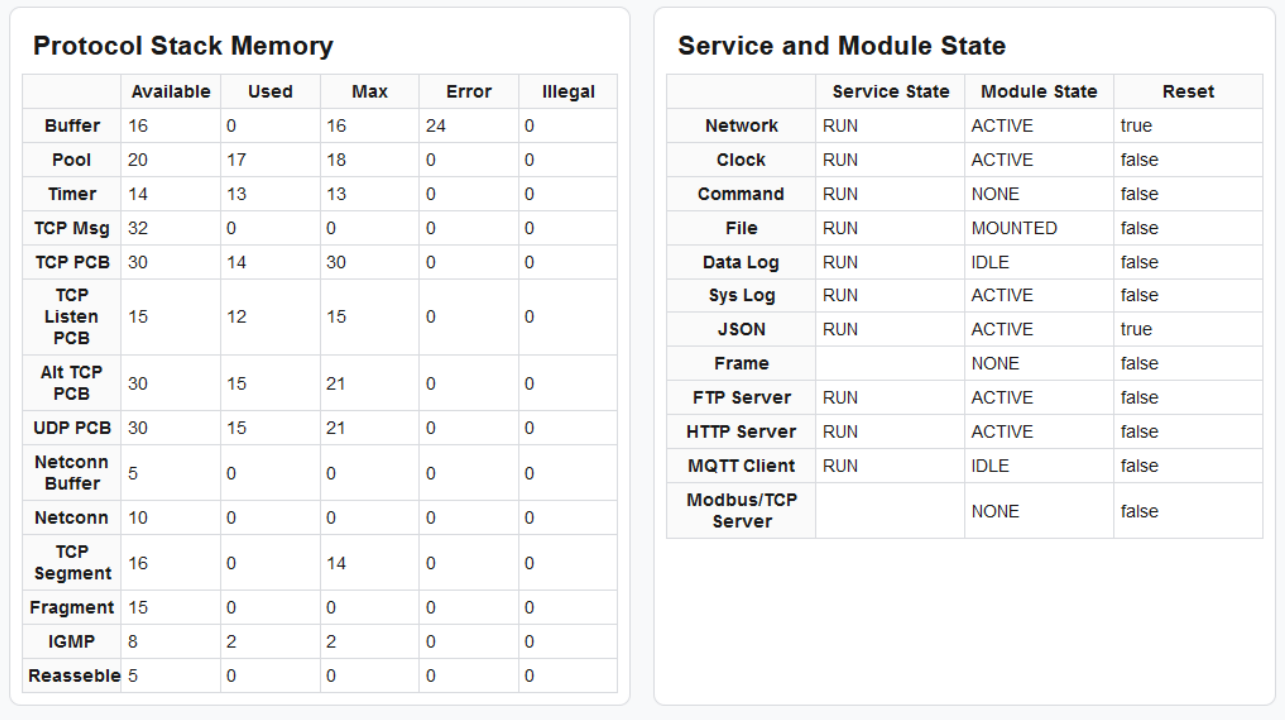
| Parameter | |
|---|---|
| Uptime | Uptime since start in seconds |
| Mem Free | Free memory in bytes |
| Mem Free Min | Minimum of free memory since start |
| Mem Error | Number of memory (allocation) errors |
| File Mem Total | Total Size of file system in kBytes |
| File Mem Free | Free memory in file system in kBytes |
| Temperature | System temperature |

**Tab. 33 System Status Memory Information**

System Identification

| Parameter | |
|---|---|
| Serial Number | Serial number of the device |
| Hardware Revision | Hardware revision of the device |
| Firmware Revision | Firmware revision of the device |
| IO Slot 0/1 Class | Function class of IO slot 0/1 |
| IO Slot 0/1 Type | Function type of IO slot 0/1 |
| IO Slot 0/1 Revision | Hardware revision of IO slot 0/1 |

**Tab. 34 System Status Identification**



**Protocol Stack Memory**

| | Available | Used | Max | Error | Illegal |
|---|---|---|---|---|---|
| Buffer | 16 | 0 | 16 | 24 | 0 |
| Pool | 20 | 17 | 18 | 0 | 0 |
| Timer | 14 | 13 | 13 | 0 | 0 |
| TCP Msg | 32 | 0 | 0 | 0 | 0 |
| TCP PCB | 30 | 14 | 30 | 0 | 0 |
| TCP Listen PCB | 15 | 12 | 15 | 0 | 0 |
| Alt TCP PCB | 30 | 15 | 21 | 0 | 0 |
| UDP PCB | 30 | 15 | 21 | 0 | 0 |
| Netconn Buffer | 5 | 0 | 0 | 0 | 0 |
| Netconn | 10 | 0 | 0 | 0 | 0 |
| TCP Segment | 16 | 0 | 14 | 0 | 0 |
| Fragment | 15 | 0 | 0 | 0 | 0 |
| IGMP | 8 | 2 | 2 | 0 | 0 |
| Reasseble | 5 | 0 | 0 | 0 | 0 |

**Service and Module State**

| | Service State | Module State | Reset |
|---|---|---|---|
| Network | RUN | ACTIVE | true |
| Clock | RUN | ACTIVE | false |
| Command | RUN | NONE | false |
| File | RUN | MOUNTED | false |
| Data Log | RUN | IDLE | false |
| Sys Log | RUN | ACTIVE | false |
| JSON | RUN | ACTIVE | true |
| Frame | | NONE | false |
| FTP Server | RUN | ACTIVE | false |
| HTTP Server | RUN | ACTIVE | false |
| MQTT Client | RUN | IDLE | false |
| Modbus/TCP Server | | NONE | false |

**Fig. 23 System Status Overview**

Protocol Stack Memory

This section shows statistic of available, currently used and maximum used memory resources of the protocol stack. The last two columns show the number of allocation errors and illegal operations.

Service and Module State

This section monitors the state of LucidIoT services. Details can be found in the sections describing the services (→ 3.3, 3.4).

In order to avoid overflows, counters, minimum and maximum values are cleared once per day after they have been written to the system log file (→ 3.4.9.2).

## 3.4.2 File System

The LucidIoT file system stores files for the following purposes in an internal memory:

- Data logging service (→ 3.4.3)
- System diagnostic logging service (→ 3.4.9)
- Customized Web Pages (→ 3.3.9.3)

The files are accessible by the FTP Server (→ 3.3.12).

## 3.4.3 Data Logging

The data logging service saves IO values and system data to the file system.

If data logging service and at least one log value entry are enabled, recording starts with configurable interval.

### 3.4.3.1 Configuration

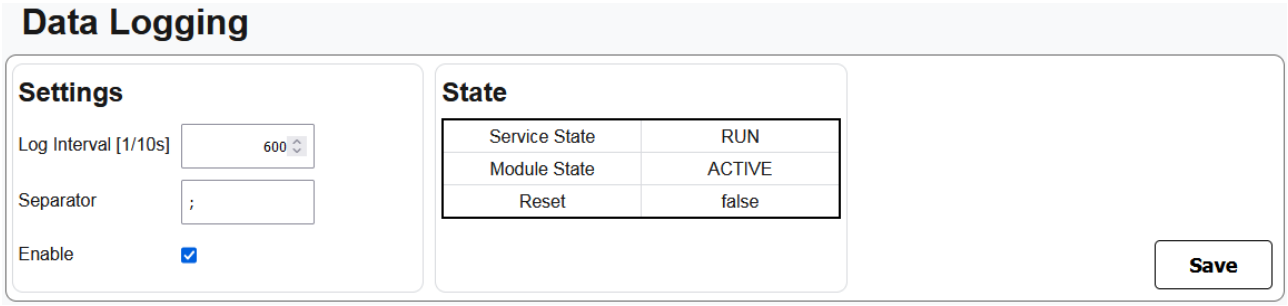



**Fig. 24 Data Logging Configuration and State Page**

Only users with admin rights are allowed to access this page. It is not visible for other users.

Data Logging Configuration

| Parameter | |
|---|---|
| Log Interval | Log interval in 1/10 seconds (Value 600 -> 60s)<br>Specifies the time between two log events. On a log event, a new data row is generated |
| Enable | If set, data logging is enabled |
| Separator | Defines character dividing columns inside a data row. |

**Tab. 35 Data Logging Configuration Parameters**

Settings are updated by clicking the *Save* button. Updates of parameters are applied immediately and do not require a restart.

### 3.4.3.2 Data Logging Values


**Fig. 25 Data Logging Value Configuration**

The data logging service periodically saves up to 16 configurable values to the data log file. Fig. 25 shows the configuration details for the first data logging value.

Data Logging Value Configuration

| Parameter | |
|---|---|
| Enable | Enables the data logging value |
| Name | Descriptive name of the data column written to the header row. |
| Value Id | Value identifier to log (→ 3.3.10.4.1). |
| Format | Value format (→ 3.3.10.4.4) |

**Tab. 36 Data Logging Value Configuration Parameters**

Settings are updated by clicking the *Save Values* button at the end of the list. Updates of log value settings are applied immediately and do not require a restart.

The auto format is used if the configured value format is not supported by the value identifier.

### 3.4.3.3 Data Log Files

The data logging service creates one data log file per day. When the system date changes, the current data log file is closed and a new file is created becoming the active data log file.

The function creates sub-folders for each year and month. Data log files have the filename pattern `YYYYMMDD.log`. They are stored in the folder `/log/YYYY/MM/`.

The header of the data log file is written, when:

- A new data log file is created.
- The system restarts while an existing data log file is opened
- Change of data logging value configuration

Each log data row starts with a time stamp column subsequently followed by the enabled log values.

The service uses the local time zone and formats the time stamp in the format `YYYY-MM-DD HH:MM:SS` using 24 hours format.

Columns are separated by the separation character.

Example:
Data Logfile /log/2021/01/20210101.log

```
Time;DigitalInSwitch;RoomTemp
2021-01-01 00:00:00;0;20.0
2021-01-01 00:00:01;0;20.1
2021-01-01 00:00:02;0;20.0
2021-01-01 00:00:03;1;19.9
2021-01-01 00:00:04;1;20.0
```

The example shows the first five seconds of recording for the 1st of January 2021.

The file starts with the header row. It contains the mandatory time stamp and the names of two active log values *DigitalInSwitch* and *RoomTemp* separated by a semicolon separation character.

The subsequent data log rows consist of the time stamp (one log event per second) and the data log values.

Inconsistent time stamp values
Time stamps of each row of the data log file are expected to be increasing continuously. Changing the system clock may lead to gaps or overlapping time stamps. This is especially relevant if the system clock is synchronized by SNTP (→ 0).

Memory overflow prevention
Logfiles are continuously added to the file system what may lead to a shortage of available memory.
The data logging service prevents this and checks if enough free memory is available before a new data log file is created. If the capacity limit is reached, the oldest data log files are deleted in a FIFO way.
The capacity limit is reached, when 75% percent of the total internal memory are used.

Example
For an internal memory size of 4 GB, 1 GB of internal memory always remains free. 3GB of internal memory are used for data logging and other files (e.g. custom web pages and system diagnostic).

### 3.4.4 System Clock

The system clock provides an accurate, network synchronize-able date and time information.

The system clock supports local time zones, DST and synchronization with a SNTP timeserver.

The system clock is backed up by a capacitor.

#### 3.4.4.1 Configuration

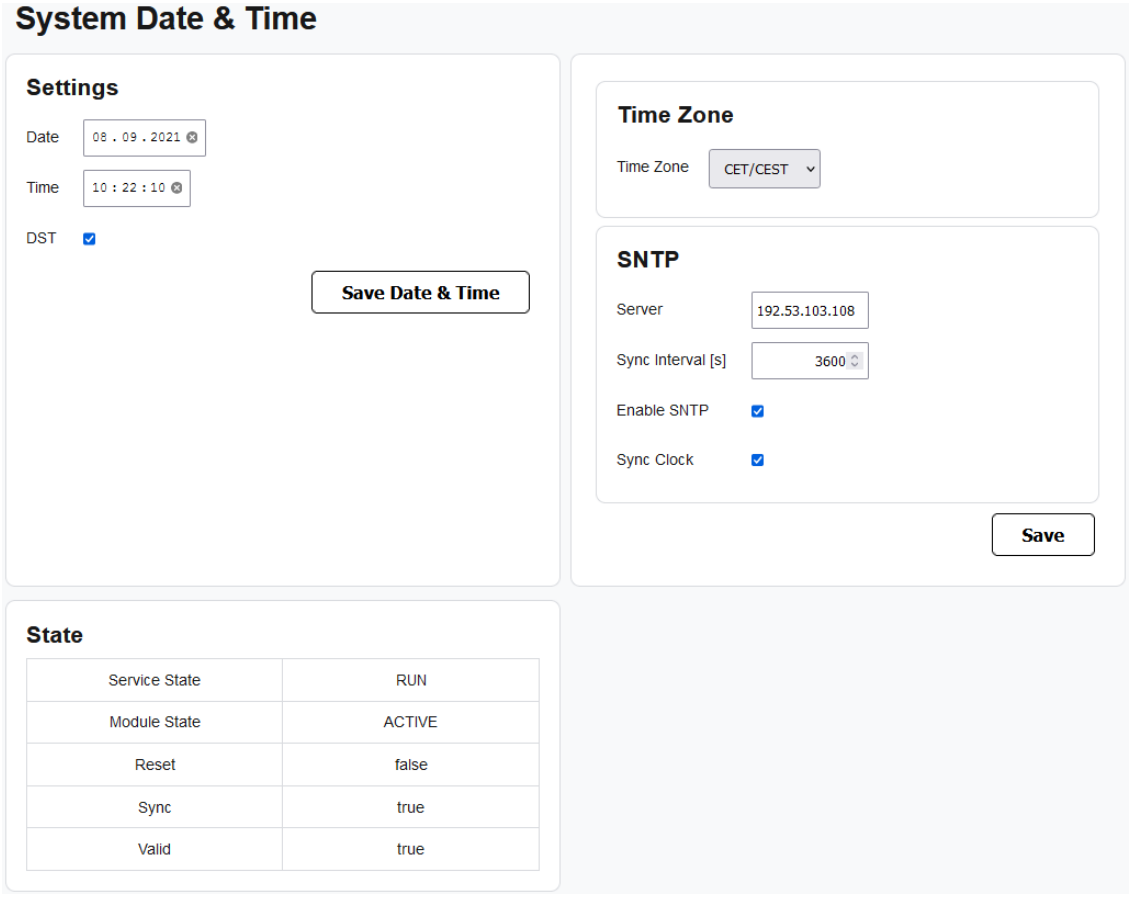



**Fig. 26 System Clock Configuration and State Page**

Fig. 26 shows configuration and status of the system clock.

Only users with admin rights are allowed to access this page. It is not visible for other users.

System Clock Settings

| Parameter | |
|---|---|
| Date | Local date |
| Time | Local time in 24 hour format |
| DST | Daylight Saving Time<br>If set, the date and time are in DST. Value is calculated internally, manual changes are ignored. |

**Tab. 37 Clock Configuration**

Clicking the button *Save Date & Time* saves date and time information. If *Enable SNTP* and *Sync Clock* are active, it is not possible to change date and time manually.

Time Zone
Selection of the local time zone.

SNTP
(→ 0)

Clicking the button *Save* updates local time zone and SNTP configuration.

## 3.4.5 User Account Management

The user account management controls and restricts access to LucidIoT's functions

| Access Right Groups | System Config | Access IO | Access IO Config | Access FTP | Read State |
|---|---|---|---|---|---|
| Enable | NO | NO | NO | NO | YES |
| FTP | NO | NO | NO | YES | YES |
| IO Value | NO | YES | NO | NO | YES |
| IO Config | NO | NO | YES | NO | YES |
| Admin | YES | YES | YES | YES | YES |

**Tab. 38 User Access Rights**

In order to access LucidIoT functions, a user needs to be logged in.
For example, the JSON interface and web interface provide functions, which are restricted and users need appropriate access rights in order to execute them.

Users can belong to one or more access right groups, users with admin access rights have full access to all functions.

Including the master admin account, 6 user accounts can be created.

User names need a minimum length of 4 characters and can be up to 8 characters long.

Passwords need a minimum length of 5 characters and can be up to 12 characters long.

### 3.4.5.1 Master Admin Account

The master admin account is a special account, which has additional rights compared to other user accounts with (standard) admin access rights.

The master admin account cannot be renamed or deleted and its access rights cannot be changed.

On a new device (or after a factory reset) only the master admin account exists.

The default login user name of the master admin account is *admin*, the default password is *admin*.

Password Considerations
It is recommended to change the password of the master admin account.
If the password got lost, it can only be resorted by resetting the device to factory defaults (→ 3.4.7).

Master Admin Account Rights
Only the master admin account has the right to write to secure memory section (→ 3.4.6.1).

#### 3.4.5.2 Login and Session

The JSON interface (→ 3.3.14, → 5.2) and the web interface make use of the user account management. Users have to log in before access to some functions is granted. Only a few basic commands can be accessed without log in.

The user log in with JSON protocol command *AccLogin* (→ 5.3.2.1) and log out with *AccLogout* (→ 5.3.2.2).

In case of successful login, *AccLogin* returns a unique 32 bit session identifier *SessionId* for the created session. Commands may require the valid *SessionId*. Otherwise, the command is denied.

If an already logged in user logs in again, the old session is released and replaced by the new session.

Inactive sessions are released and the user is logged out after a timeout of 10 minutes.

### 3.4.5.3 Configuration

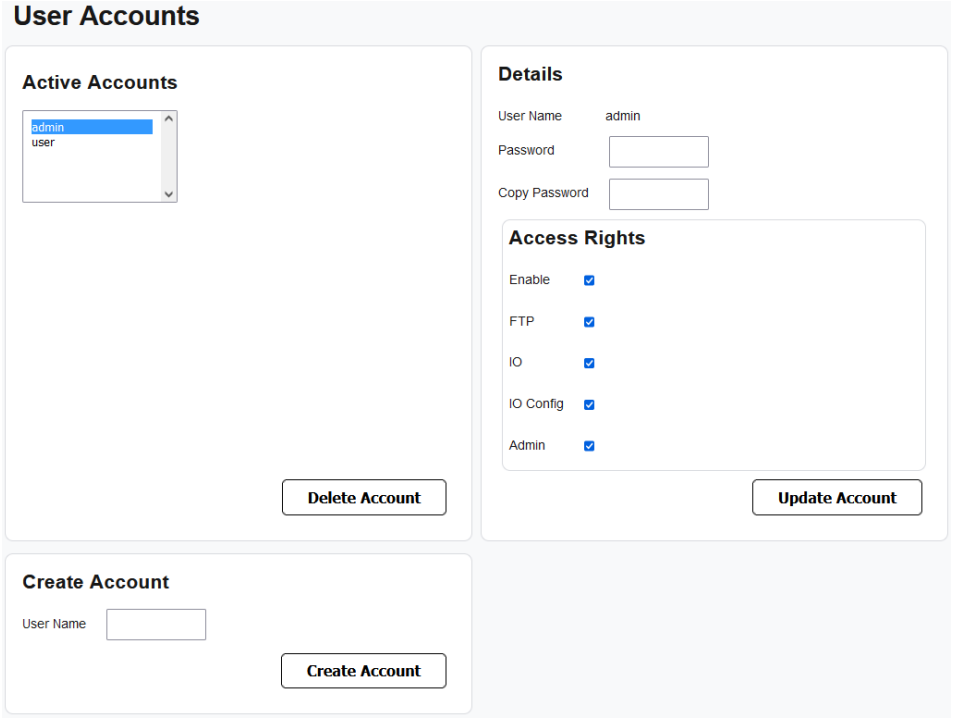



**Fig. 27 User Account Configuration**

Fig. 27 shows the user account configuration page. Admin rights are necessary in order to access page.

Only users with admin rights are allowed to access this page. It is not visible for other users.

*Active Accounts* lists all created user accounts.

When a user account is selected, details are loaded. Access rights and password can be changed and stored by clicking the *Update Account* button.

Changing password of master admin accounts requires the current password in the *Copy Password* filed.

The account selected in *Active Accounts* list is deleted by clicking *Delete Account* button.

A new user account can be created by entering the new *User Name* and clicking onto *Create Account*. The created account is empty, password and access rights must be configured separately.

## 3.4.6 Security

LucidIoT interfaces (e.g. HTTP, MQTT, JSON, FTP) can be protected by TLS communication.

### 3.4.6.1 Secure Memory

The secure memory is a memory section of the microcontroller providing 8 locations where X.509 certificates and private keys can be stored.

Data can be written to the secure memory by the master admin account only (→ 3.4.5.1).

Private data cannot be read from secure memory. For identification purpose, public data of a certificate can be read by the master admin account.

Interfaces, which are protected by TLS, can refer to the secure memory by the secure memory identifier parameters *Certificate Id*, *CA Id* and *Private Key Id*.

#### 3.4.6.1.1 Managing Secure Memory

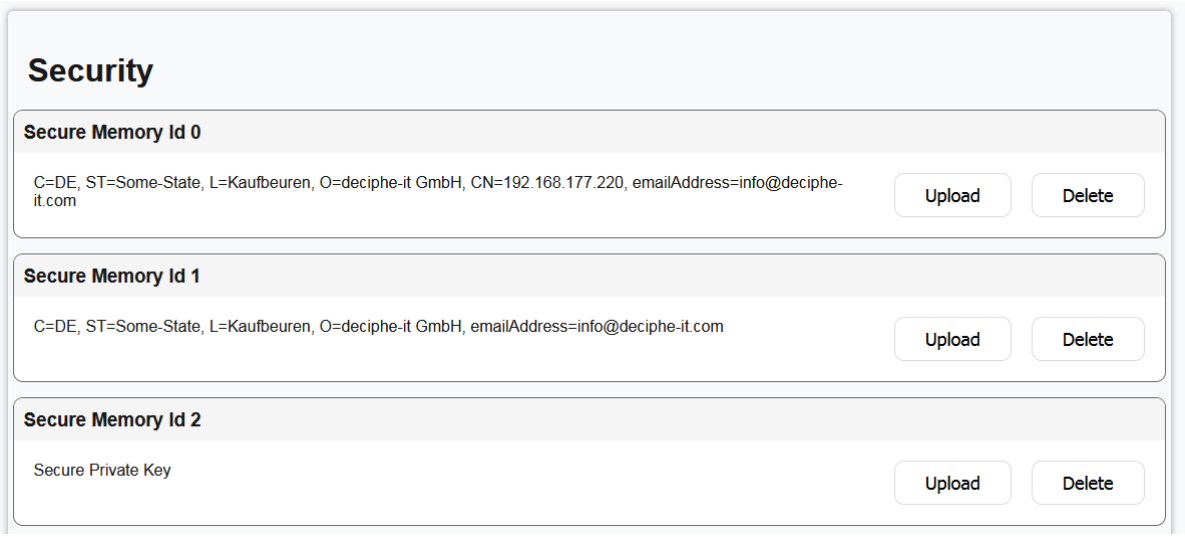



**Fig. 28 Secure Memory Web Interface Configuration**

Fig. 28 shows the secure memory configuration page.

Only the master admin is allowed to access this page. It is not visible for other users.

The page shows the type of data is stored in the secure memory locations. Only public data is shown.

The button *Upload* allows the user to upload a new certificate or private key from the local computer to the secure memory location.

The button *Delete* clears a secure memory location and makes it invalid.

Note

If a certificate or private key is used by an interface, it must not be deleted. Otherwise, the interface becomes inaccessible.

#### 3.4.6.2 TLS / Certificates

This section describes by a simple example how to create a Certificate Authority and issue a certificate which can be used in LucidIoT. OpenSSL (https://www.openssl.org/) is used for this purpose.

LucidIoT needs certificates and private keys in Base64 (PEM) format. LucidIoT has been tested with 2048 bit RSA certificates.

Certificates are a complex topic and it is beyond this document to explain it in detail.

##### 3.4.6.2.1 Create a CA (Certificate Authority)

A certificate is issued by a Certificate Authority and at first a CA needs to be set up.

OpenSSL creates a self-signed CA certificate with the following commands:

Create a private key for the CA

```
openssl genrsa -out ca-key.pem 2048
```

This command creates an RSA private key with 2048 bit which will be stored in the file ca-key.pem
This key must be kept secret and must never be published and never be uploaded to LucidIoT.
The private key can be password protected. In this case it must be provided when issuing certificates.
For simplicity reasons, the private key of the CA is not encrypted in this example.

Create CA root certificate

```
openssl req -x509 -new -days 1000 -nodes -extensions v3_ca -key ca-key.pem -out ca-root.pem -sha512
```

This command creates a CA root certificate ca-root.pem with a validity of 1000 days which can be uploaded to LucidIoT Secure Memory.

##### 3.4.6.2.2 Issue a Certificate

A LucidIoT device identify itself by a certificate. The certificate can be issued by the CA created in →3.4.6.2.1.

The following commands create a certificate and a private key, which can be uploaded to LucidIoT Secure Memory.

Create a private key

```
openssl genrsa -out cert-key.pem 2048
```

This command creates an RSA private key with 2048 bit.
The private key can be found in the file cert-key.pem. No password must be set for the private key because LucidIoT supports unencrypted private keys only.

Create a CSR (Certificate Signing Request)

```
openssl req -new -key cert-key.pem -out cert.csr -sha512
```

The CSR is created for an entity with a Common Name. This should be either the device name or its IP address.

Issue a certificate by the CA

```
openssl x509 -req -in cert.csr -CA ca-root.pem -CAkey ca-key.pem
-CAcreateserial -out cert-pub.pem -days 365 -sha512
```

The issued certificate can be found in cert-pub.pem. It has a validity of 365 days.

If the private key of the CA is encrypted, the password must be provided.

Beside of the mandatory Common Name (CN) field, also the Subject Alternative Name (SAN) extension may be required for a connection.

### 3.4.7 Maintenance Mode and Recovery

The maintenance mode allows the user to access functions, which are not accessible in normal operation mode.

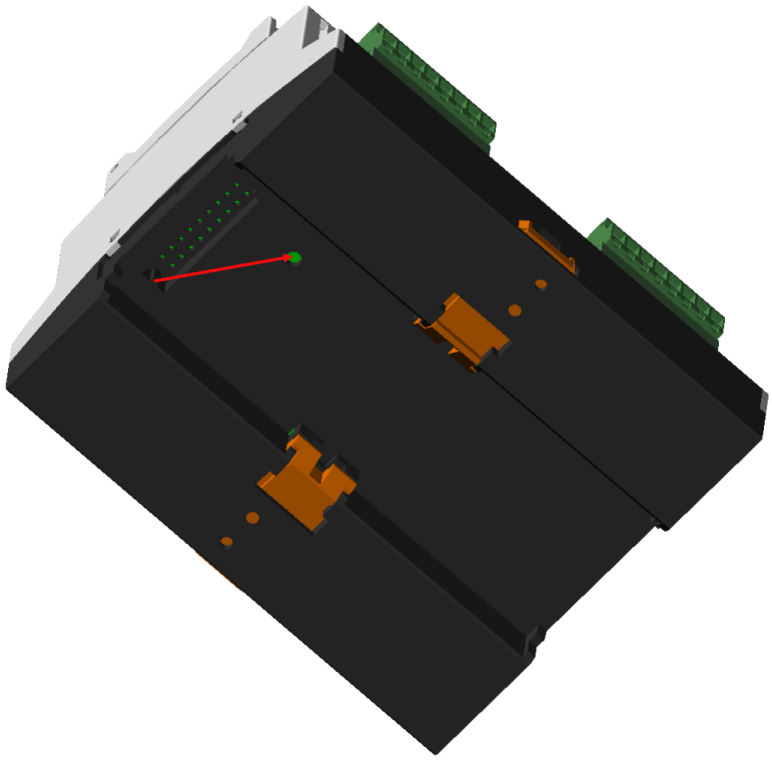


**Fig. 29 Maintenance Switch**

In maintenance mode, LucidIoT operates with a minimum of functionality with default configuration.

The main purpose of the maintenance mode is to recover LucidIoT by resetting it to factory configuration.

If for example, the a wrong network configuration makes LucidIoT unreachable, maintenance mode can be used to start the device with default configuration and reset it to factory configuration.

Entering Maintenance Mode

The Maintenance Mode is entered by the following steps:

1. Remove power supply and all IO terminal connectors from the device.
2. Connect the device directly to a computer by an Ethernet cable.
3. Press and hold the maintenance push button located at the backside of the device (Fig. 29) while connecting the power supply.
4. The red *State LED* blinks once maintenance mode was entered
5. LucidIoT is now reachable by the web interface by http://192.168.177.240/login.htm

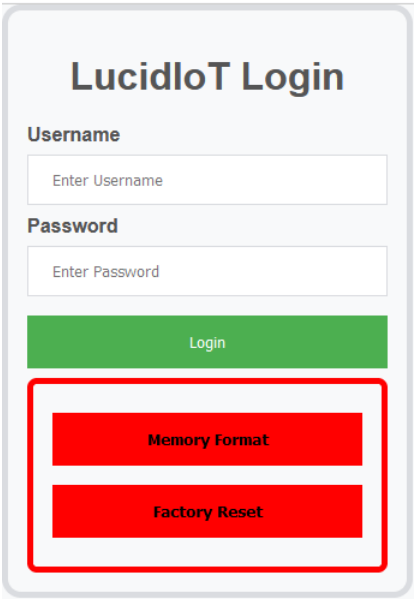



**Fig. 30 Maintenance Mode Login**

Maintenance Mode Functionality

Fig. 30 shows the login page in maintenance mode.

Only the master admin account is allowed to login.

Clicking the button *Memory Format* formats the internal memory. All stored data e.g. log files, custom web pages are deleted. The red *State LED* blinks while format process is ongoing. After format completed, the power supply should be recycled. Configuration values are preserved.

Clicking the button *Factory Reset* sets all configuration values (but not the internal memory) to factory defaults and restarts the device. All configuration e.g. user accounts, secure memory are deleted.

For memory format and factory reset, no username or password are needed.

LucidIoT operates In maintenance mode with the following configuration:

- TLS is disabled for all services, default IPv4 address 192.168.177.240
- JSON Interface is operating on port 50000
- Web interface running on port 80
- All other services and applications are disabled

### 3.4.8 Firmware Update

The firmware of LucidIoT can be updated. For IoT devices this is important as they may be exposed to the Internet and new security problems arising e.g. by attacks or by security breaks must be fixed quickly.

#### 3.4.8.1 Update Process

Firmware updates have the file name format LIOxxxxx.bin, where xxxxx is replaced by the 5 digit revision number.

The Firmware Update Tool allows uploading of the new firmware file by FTP.

During the update process the signals applied to the terminals X1 – X4 may be outside the specified limits and the terminal connectors must be disconnected during the update process.

### 3.4.9 System Diagnostic

The system diagnostic function records system messages and statistics in the system diagnostic file. Information is stored in human readable text format.

If a LucidIoT service reports an error, the *Service State* or *Module State* indicates this. More details about the error can be found in the system diagnostic files.

Example

There are many reasons, why the MQTT client is not able to connect to a MQTT server. In this case, the MQTT *Module State* is set to CONNECT_PND (→ 3.3.10.3).
It is e.g. possible that the certificate configuration is invalid or that the DNS server was not able to resolve the MQTT server address. These details can be found in the system diagnostic log files.

#### 3.4.9.1 System Diagnostic Data

The system diagnostic data is saved to the file system and is accessible by the FTP server.

Files are stored in the \SYSLOG folder. The filename starts with the system clock date, followed by *.slg file extension.

A new file is created for each day on midnight.
The last 90 files are available. If this limit is reached, the oldest file is removed before a new one is created.

```
2021-03-30 10:46:10 | SUC: MQTT, DNS resolved, server DEC-PC-1, IP 192.168.177.21
2021-03-30 10:46:10 | INF: LIoMqttConnect, connect
2021-03-30 10:46:29 | WRN: MQTT, disconnected
2021-03-30 10:46:29 | SUC: MQTT, DNS resolved, server DEC-PC-1, IP 192.168.177.21
2021-03-30 10:46:29 | INF: LIoMqttConnect, connect
2021-03-30 10:46:47 | WRN: MQTT, disconnected
```

**Tab. 39 System Diagnostic Data Example**

| System Diag. Class | |
|---|---|
| SUC | Inform about success |
| INF | General information |

| | |
|---|---|
| WRN | Warning<br>A warning indicates that a function failed because of an error that was recoverable. |
| ERR | Error<br>An error indicates that a function failed because of a critical error. |

**Tab. 40 System Diagnostic Classes**

Tab. 39 shows a few lines belonging to the MQTT client connection process. Each line of the file represents a system diagnostic log entry. The entry starts with a timestamp, followed by the system diagnostic class (Tab. 40). The log message starts identifying the log source, which created reported the message, followed by the descriptive log message text.

The first line of the example shows that LucidIoT successfully resolved the IP of the MQTT server name. The second line shows that MQTT client tries to connect to the server and the 3rd line indicates with a warning that the connection could not be established. The following lines show that subsequent connection tries failed, too. The MQTT server is most likely unreachable.

When LucidIoT is started, it tries to open an existing file corresponding with current system clock information, or to create a new file. If the system clock is invalid on start (e.g. because it could not be restored) the first messages are stored in the default system log file 20160101.slg. Once the clock information becomes valid, the appropriate system log file is used.

The system diagnostic file of the current day can be inspected via the web interface.

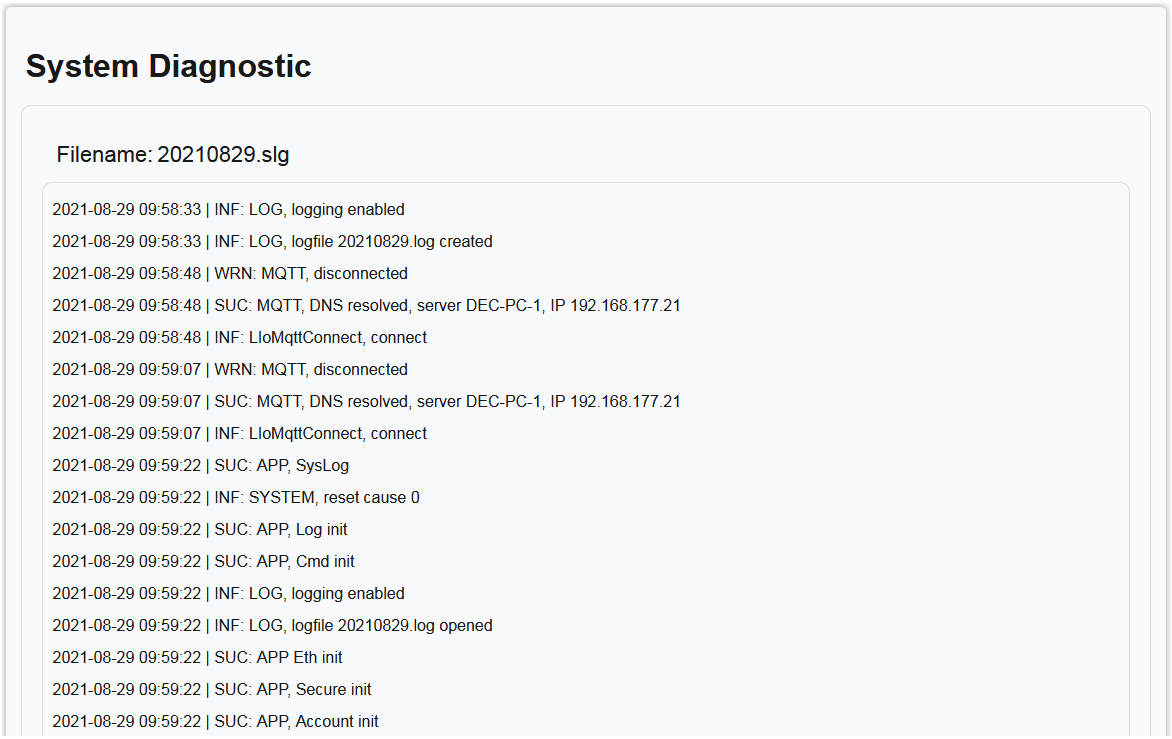



**Fig. 31 System Diagnostic Web Interface**

### 3.4.9.2 Save Daily Diagnostic Information

Before system diagnostic file is closed, status of services is written to the system diagnostic file. Incrementing status variables (e.g. packet counters) and minimum maximum limits (e.g. minimum free memory) are reset after logging.

# 4 IO Operation

This section describes the IO processing functionality. The relevant parts of this section depend on the product configuration.

## 4.1 IO Slot Concept and Connections

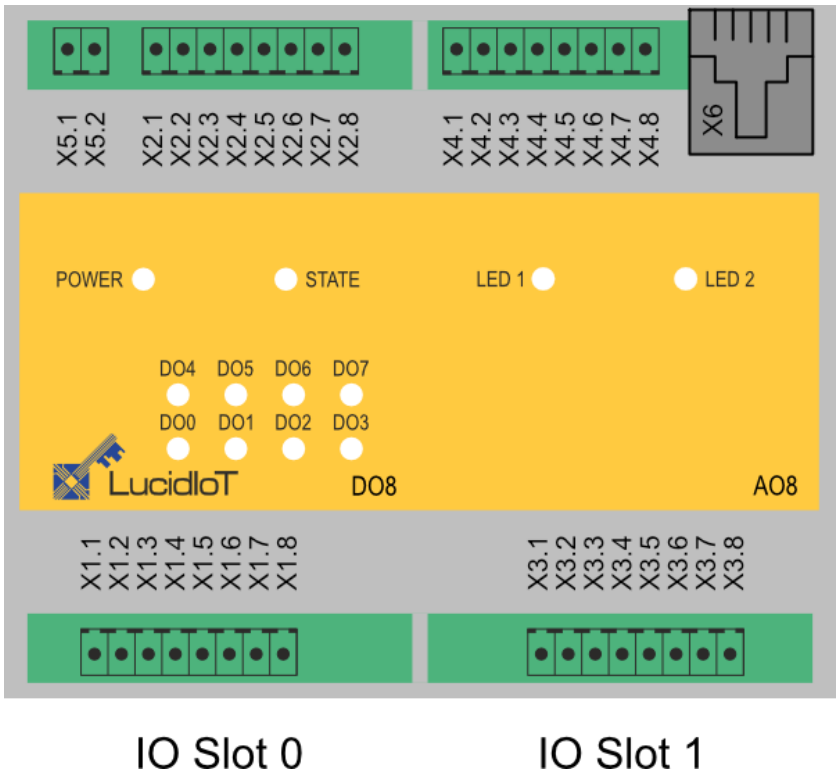



**Fig. 32 LucidIoT Slot Concept**

Fig. 32 shows the slot concept of LucidIoT. The IO Slot 0 (left) and IO Slot 1 (right) are working independently with configurable functionality. This example shows the LIOT-DO8-I-AO8-10V module.

IO Slot 0 is configured as DO8 function class with 8 digital solid state relay outputs.
IO Slot 1 is configured as AO8 function class with 8 analog 0-10V output channels.

Function classes define the general function of the IO Slot (e.g. DO8 for 8 digital output channels). The function class type defines the function in detail (e.g. I for digital outputs with solid state relays).

The terminals X1 and X2 are connected to IO Slot 0.
The terminals X3 and X4 are connected to IO Slot 1.

The functionality of the function classes is explained in the section 4.2.

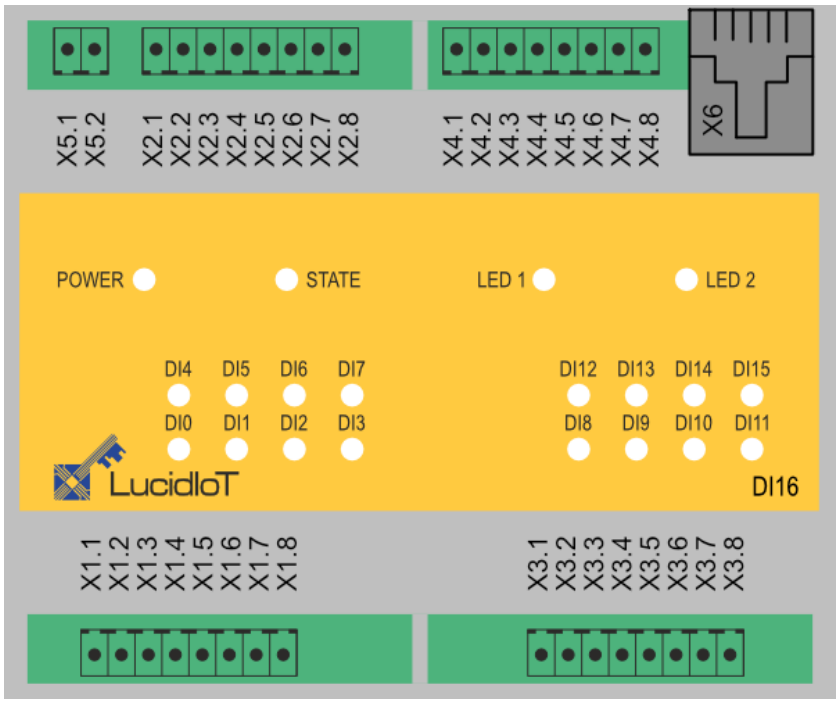



**Fig. 33 LucidIoT with Function Classes DI8 and DI8**

Fig. 33 shows the configuration with two identical function classes like LIOT-DI8-24V-DI8-5V with 8 24V digital inputs and 8 5V digital inputs.

Configurations with same function classes and function class types are labeled as DI16, DO16, AI16, AO16 or RI16. The IO channels of IO Slot 1 refer to the upper 8 channels.

### 4.1.1 IO Channel Numbers

Each IO channel is identified by an IO channel number starting with 0, referring to the first IO channel of IO Slot 0.

For example, the product LIOT-DO8-I-AO8-10V, IO channel number 0 refers to the first digital output channel, IO channel number 7 to the last digital output channel.

IO channel number 8 is linked to the first analog output channel on IO Slot 1. The device has in total number of 16 channels.

A maximum of 16 IO channels is supported and 15 is the highest IO channel number.

## 4.2 IO Function Classes

### 4.2.1 Digital Input (DI8 Function Class)

The DI8 function class acquires the states of 8 digital input signals.

| Function Class | Value | Channels |
|---|---|---|
| DI8 | 0x0010 | 8 |

**Tab. 41 Digital Input Function Classes**

| Function Class Type | Value | Threshold Level |
|---|---|---|
| 5 | 0x1000 | 5V |
| 10 | 0x1001 | 10V |
| 24 | 0x1005 | 24V |

**Tab. 42 Digital Input Function Class Types**

Tab. 41 and Tab. 42 list the DI8 function class and it's supported function class types.

#### 4.2.1.1 IO Signal

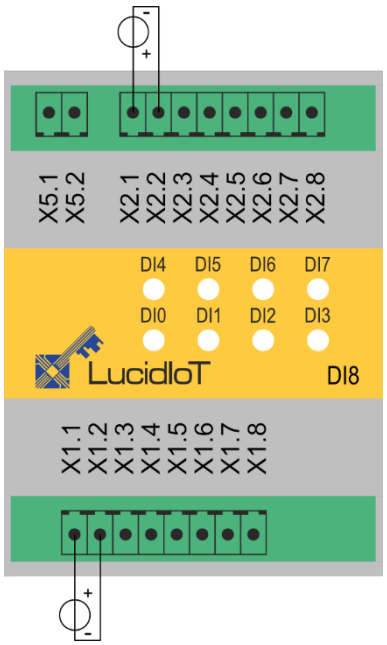



**Fig. 34 DI8 at IO Slot 0**

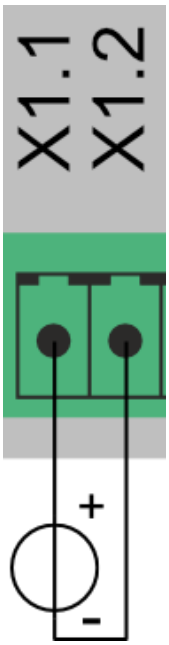



**Fig. 35 DI8 Signal**

Fig. 34 shows the IO terminals of the DI8 function class installed at IO Slot 0.

Two voltage sources are connected to X1.1 and X1.2 (DI0) and X2.1 and X2.2 (DI4).

Fig. 35 shows the voltage source connected to the IO terminals X1.1 and X1.2 (DI0) in detail.

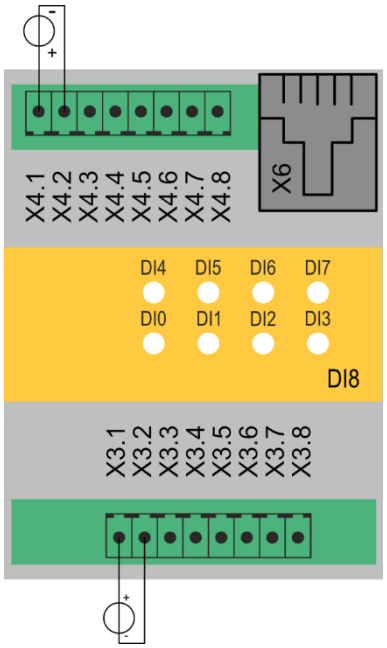



Fig. 36 shows the IO terminals of the DI8 function class installed at IO Slot 1.

Two voltage sources are connected to X3.1 and X3.2 (DI0) and X4.1 and X4.2 (DI4).

**Fig. 36 DI8 at IO Slot 1**

The IO terminals of the DI8 function class and their IO channel numbers are listed in Tab. 43.

A high input value is indicated by a green LED on the front panel.

| IO Terminal | IO Slot | Signal | IO Channel Number |
|---|---|---|---|
| X1.1 | 0 | DI0 + | 0 |
| X1.2 | | DI0 - | |
| X1.3 | | DI1 + | 1 |
| X1.4 | | DI1 - | |
| X1.5 | | DI2 + | 2 |
| X1.6 | | DI2 - | |
| X1.7 | | DI3 + | 3 |
| X1.8 | | DI3 - | |
| X2 | | DI4 ... DI7 | 4 – 7 |
| X3 | 1 | DI0 ... DI3 | 8 – 11 |
| X4 | | DI4 ... DI7 | 12 – 15 |

**Tab. 43 DI8 IO Terminal Connector**

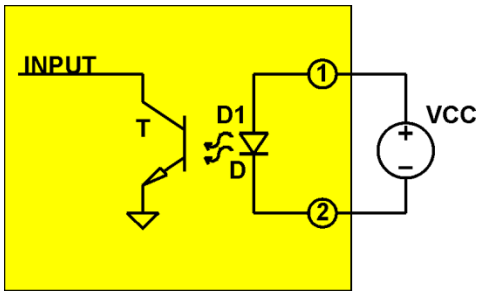



Fig. 37 illustrates the principle of the digital input channels. The input signal applied to the IO terminals 1 (e.g. X1.1) and 2 (e.g. X1.2) powers the LED of the opto-coupler, which insulates the input signal and the acquisition hardware.

**Fig. 37 Principle of Digital Input**

All inputs are floating and do not have any common connections e.g. to ground signal.

Never apply voltages higher than 30V (or lower than -30V) to any IO terminal. This would damage the device. Consider $V_{HighMax}$, also.

#### 4.2.1.1.1 Threshold Level

The digital inputs are available for different threshold levels:

| Threshold Level | $V_{LowMax}$ | $V_{HighMin}$ | $V_{HighMax}$ |
|---|---|---|---|
| 5 V | 2.5 V | 3.5 V | 7.5V |
| 10 V | 6.0 V | 8.5V | 15V |
| 24 V | 16.0 V | 21.0 V | 30V |

**Tab. 44 Digital Input Channel Signal Threshold Levels**

Tab. 44 lists the characteristic voltages for the different threshold levels. Voltages below $V_{LowMax}$ result in a LOW value, voltages higher than $V_{HighMin}$ in a HIGH value. Because of the input hysteresis, voltages between $V_{LowMax}$ and $V_{HighMin}$ do not change the logic value.

In order to prevent excessive stress on the digital inputs, the maximum applied voltage must not exceed $V_{HighMax}$.

A high input state is indicated by a green status LED.

Example
When interfacing a 24 V signal, the applied voltage of a HIGH state must be higher than 21.0 V. The voltage of a low state needs to be lower than 16.0 V. For the voltages between, the last detected stable value remains.

#### 4.2.1.1.2 Real-time Considerations

Operating systems for personal computers are not made for deterministic real-time operation. Because of multitasking it cannot be ensured that a task will continue to run within a specified interval.

Ethernet is also no real-time bus and limits the timing.

Assuming that short pulses (e.g. shorter than 10 ms) should be detected, the computer has to read the input value at least 100 times per second what is not realistic. It is possible that a pulse is located between two readings and it would be missed.

Edge detection and count modes improve the real-time characteristics.

#### 4.2.1.1.3 Filter

The digital input signal is filtered and only stable high or low signals are evaluated. This function allows the input channel to debounce the signal.

Signals are only considered valid if they are stable during the time $T_{Scan}$ (→ 4.2.1.3.4, → Fig. 40).

#### 4.2.1.1.4 Input Signal Value Inversion

Digital input channels consist of input signal value and a logical input value. The input signal value is represented by the voltage applied to the input channel. The logical value is evaluated by the input processing.

In case of inversion is disabled, the input signal values and logical values are identical.

In case of input inversion is enabled by configuration parameter *inDiInverted* set to "on", the logical value is the inverted input signal value. This means that a voltage higher than $V_{HighMin}$ results in a HIGH input signal value but a LOW logical input value (→Fig. 38, Fig. 39).

All input modes support the inversion of input value.

### 4.2.1.2 Operation Modes

Each digital input channel can work in one of the following modes:

- Reflect mode
- Rising edge mode
- Falling edge mode
- Count mode

In all operation modes the input values are captured and evaluated after a stable signal has been detected.

Each digital output channel operates according to its configuration parameters (→ 4.2.1.3).

#### 4.2.1.2.1.1 Reflect Mode

The reflect mode captures the state of the input signal.

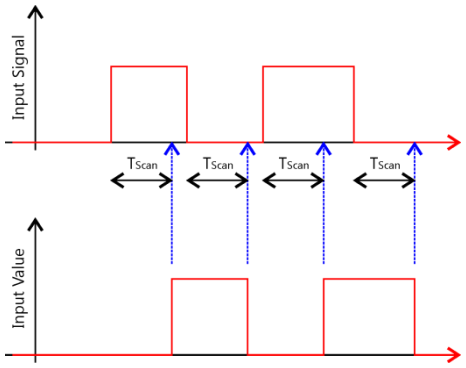



Fig. 38 illustrates the processing of digital input channels in reflect mode.

The IO channel value is captured after the rising edge of the input signal has been detected and remained stable over the interval $T_{Scan}$.

**Fig. 38 Digital Inputs in Reflect Mode**

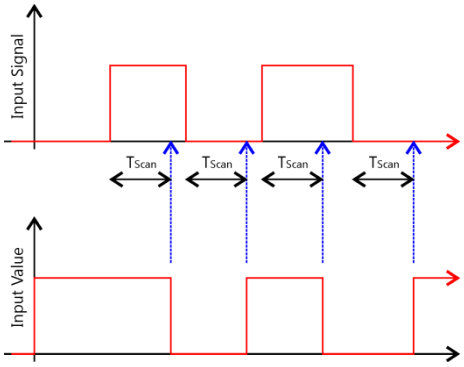
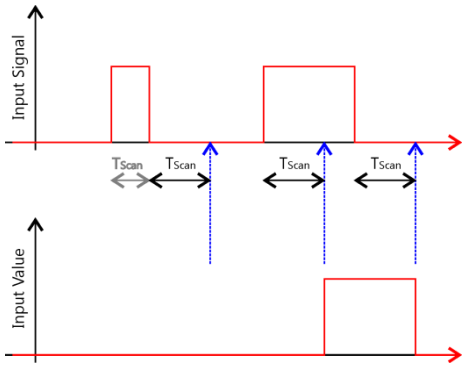



Fig. 39 illustrates the digital input signal and the inverted input value with *inDiInverted* set to on.

**Fig. 39 Inverted Digital Inputs**




In the case that a pulse of the input signal is shorter than $T_{Scan}$, the pulse is not valid. This scenario is shown in the first pulse of Fig. 40. The first rising edge of the input signal starts the scan timer the falling edge cancels it (indicated by the gray $T_{Scan}$ interval).

Since the second pulse is longer than $T_{Scan}$ it is evaluated as valid value.

**Fig. 40 Digital Inputs Pulse Width**

Filtering digital signals and validating their stability can be used in order to surpress invalid signals (debouncing) and to make the recognition of digital input signals more reliable.

The scan interval $T_{Scan}$ is configured by the parameter *inDiScanTime* (→ 4.2.1.3.4).

#### 4.2.1.2.1.2 Edge Detection

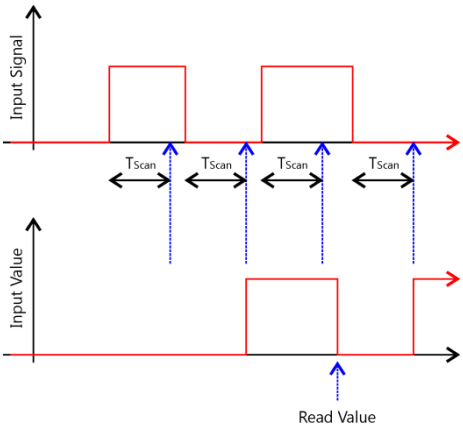



**Fig. 41 Digital Input Rising Edge Mode**

Digital input channels can operate in edge detection modes. In rising edge mode, the channel is sensitive for low-to-high transitions, in falling edge mode it recognizes high-to-low transitions.

Fig. 41 shows a digital input signal and the corresponding input value in rising edge mode. After the HIGH value was detected as being valid, the input value remains pending until it was read by the host computer.

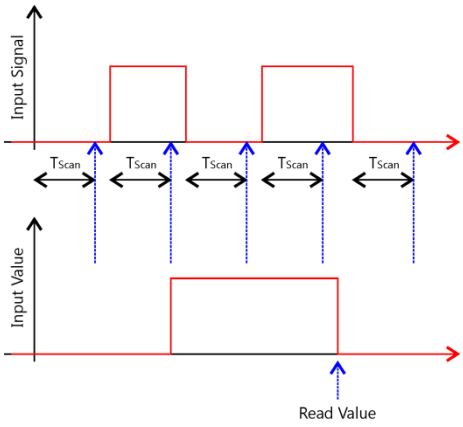



**Fig. 42 Digital Input Falling Edge Mode**

The falling edge mode (Fig. 42) operates similar as the riding edge mode does, but triggers for high-to-low transitions.

The edge modes allow detecting low-to-high transitions of the input signal without the host computer being involved.

If a transition was detected between two IO channel value readings, 1 is returned. Otherwise, 0 is returned.

#### 4.2.1.2.1.3 Count Mode

The count mode accumulates valid input pulses within a counting time specified by $T_{Count}$.

Reading the IO channel value returns the number of accumulated pulses.

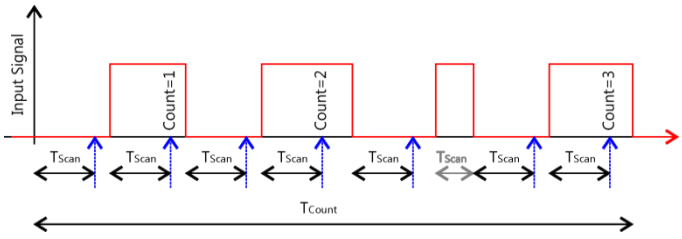



**Fig. 43 Digital Input Count Mode**

Fig. 43 illustrates a typical periodical input signal. In count mode, all valid pulses are accumulated until the counting interval $T_{Count}$ finishes.

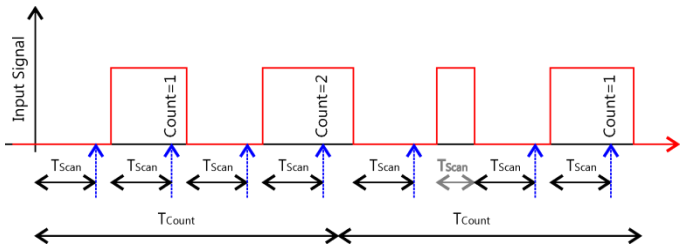



**Fig. 44 Digital Input Count Mode Short Interval**

Fig. 44 shows the same input signal, but with a shorter counting interval.

Count time has a granularity of scan time. The new counter value becomes valid when when $T_{Count}$ and $T_{Scan}$ passed.

An input signal is considered as valid when it was stable for at least the scan time $T_{Scan}$. In count mode, only stable pulses are accumulated (see the gray interrupted $T_{Scan}$ interval in Fig. 43).

Scan time starts when the rising edge of the input signal is detected. If the parameter *inDiInverted* is set to “on”, time $T_{Scan}$ starts when the falling edge of the input signal is detected.

#### 4.2.1.2.1.3.1 Count Mode Options

The counter value behavior can be controlled by the options *inDiAddCounter* and *inDiResetCounterOnRead*.

In the case *inDiAddCounter* is set to “off”, the counter value is overwritten after count time $T_{Count}$ passed and might be lost.

In the case *inDiAddCounter* is set to “on”, the counted pulses are added to the counter value after count time $T_{Count}$ passed.

The option *inDiResetCounterOnRead* controls how the counter value is changed when it was read by the host.

In the case *inDiResetCounterOnRead is set to* "*off*", reading the counter value does not change it.

In the case *inDiResetCounterOnRead* is set to “on”, the counter value is reset after reading it.

In order to avoid overflows *inDiAddCounter* ”on” should be combined with *inDiResetCounterOnRead* "on" parameter.

The option *inDiResetCounterOnRead* has only an effect together with *inDiAddCounter* set to “on”.

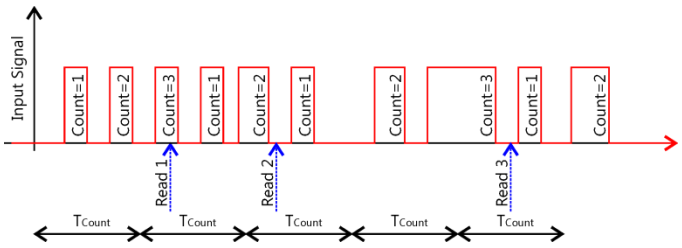



Fig. 45 shows a non-periodic input signal with 10 pulses in total. It shows 3 times where the counter value is read. In the following it is explained how the options affect the counter value.

**Fig. 45 Digital Input Counter Options**

The number of "Count" in the diagram refers to the internal counter, which is accumulated during $T_{Count}$ time.

On Read 1 the returned counter value is 2 since the internal counter value when $T_{Count}$ ended was 2. The counter value is reset on reading and 1 pulse is carried over to the next $T_{Count}$ time.

On Read 2 the returned counter value is 3 because of the internal counter value 2 plus the carried over counter 1 from the last count interval. The counter value is reset on read.

At the reading 3 the read value is 3. The counter value is reset on read.

Count mode with *inDiAddCounter* = “on”

In the case that *inDiAddCounter* is “on” and *inDiResetCounterOnRead* is “off” the pulses are accumulated but the counter is not reset on reading the value. This causes the counter value will be updated when count interval has finished and the current counter is added to the last counter value. This may result in an overflow when the counter value rolls over its maximum value of 65535.

Tab. 45 explains the different count value results of the previous example for all options.

| Mode | Value Read 1 | Value Read 2 | Value Read 3 |
|---|---|---|---|
| *inDiAddCounter* = “on” *inDiResetCounterOnRead* = “on” | 2 | 3 | 3 |
| *inDiAddCounter* = “on” *inDiResetCounterOnRead* = “off” | 2 | 5 | 8 |
| *inDiAddCounter* = “off” *inDiResetCounterOnRead* = “off” (Standard Count mode) | 2 | 3 | 2 |

**Tab. 45 Digital Input Count Mode Options**

The most useful configuration is *inDiAddCounter* and *inDiResetCounterOnRead* are set to "on". This prevents counter overflow and ensures that no pulses are lost.

#### 4.2.1.3 Configuration Parameters

##### 4.2.1.3.1 inDiValue

In reflect mode and edge detection mode IO configuration parameter *inDiValue* contains the IO channel value.

If the input is configured in count mode, *inDiValue* is 0.

| **Parameter** | *inDiValue* | **Access** | Read |
|---|---|---|---|
| **Values** | IO Value | | |
| **Default Value** | 0x00 | **Parameter Type** | 1 Byte unsigned |

**Tab. 46 IO Channel Configuration Parameter inDiValue**

##### 4.2.1.3.2 inDiMode

The IO configuration parameter *inDiMode* specifies the operation mode (→ 4.2.1.2) of the digital input channel.

| **Parameter** | *inDiMode* | **Access** | Read / Write |
|---|---|---|---|
| **Values** | Input mode<br>inactive, reflect, rising edge, falling edge, count | | |
| **Default Value** | reflect | **Parameter Type** | - |

**Tab. 47 IO Channel Configuration Parameter inDiMode**

##### 4.2.1.3.3 inDiFlags

The IO configuration parameter *inDiFlags* specifies operation flags of the digital input channel.

###### 4.2.1.3.3.1 inDiInverted

The IO configuration parameter bit *inDiInverted* specifies if the IO channel value is inverted (→ 4.2.1.1.4).

| **Parameter** | *inDiInverted* | **Access** | Read / Write |
|---|---|---|---|
| **Values** | on/off | | |
| **Default Value** | off | **Parameter Type** | 1 Bit |

**Tab. 48 IO Channel Configuration Parameter Bit inDiInverted**

###### 4.2.1.3.3.2 inDiAddCounter

The IO configuration parameter bit *inDiAddCounter* controls the IO channel value accumulation in count mode. (→ 4.2.1.2.1.3.1)

| **Parameter** | *inDiAddCounter* | **Access** | Read / Write |
|---|---|---|---|
| **Values** | on/off | | |
| **Default Value** | off | **Parameter Type** | 1 Bit |

**Tab. 49 IO Channel Configuration Parameter Bit inDiAddCounter**

#### 4.2.1.3.3.3 inDiResetCounterOnRead

The IO configuration parameter bit *inDiResetCounterOnRead* controls the IO channel value behavior in count mode when IO value is read. (→ 4.2.1.2.1.3.1)

| **Parameter** | *inDiResetCounterOnRead* | **Access** | Read / Write |
|---|---|---|---|
| **Values** | on/off | | |
| **Default Value** | off | **Parameter Type** | 1 Bit |

**Tab. 50 IO Channel Configuration Parameter Bit inDiResetCounterOnRead**

### 4.2.1.3.4 inDiScanTime

The IO configuration parameter *inDiScanTime* specifies the time $T_{Scan}$ in microseconds.

| **Parameter** | *inDiScanTime* | **Access** | Read / Write |
|---|---|---|---|
| **Values** | $T_{Scan}$ in µs<br>80 µs ≤ $T_{Scan}$ ≤ 1 s | | |
| **Default Value** | 50,000 (50 ms) | **Parameter Type** | 4 Bytes unsigned |

**Tab. 51 IO Channel Configuration Parameter inDiScanTime**

### 4.2.1.3.5 inDiCountTime

The IO configuration parameter *inDiCountTime* specifies the $T_{Count}$ time in microseconds. It is used in count mode (→ 4.2.1.2.1.3).
The value should be multiples of $T_{Scan}$.

| **Parameter** | *inDiCountTime* | **Access** | Read / Write |
|---|---|---|---|
| **Values** | $T_{Count}$ in µs<br>1 ms ≤ $T_{Count}$ ≤ 1 h | | |
| **Default Value** | 5000000 (5 s) | **Parameter Type** | 4 Bytes unsigned |

**Tab. 52 IO Channel Configuration Parameter inDiCountTime**

### 4.2.1.4 Web Interface

This section explains how the DI8 IO channel value and configuration are accessed using the web interface.

#### 4.2.1.4.1 IO Channel Value

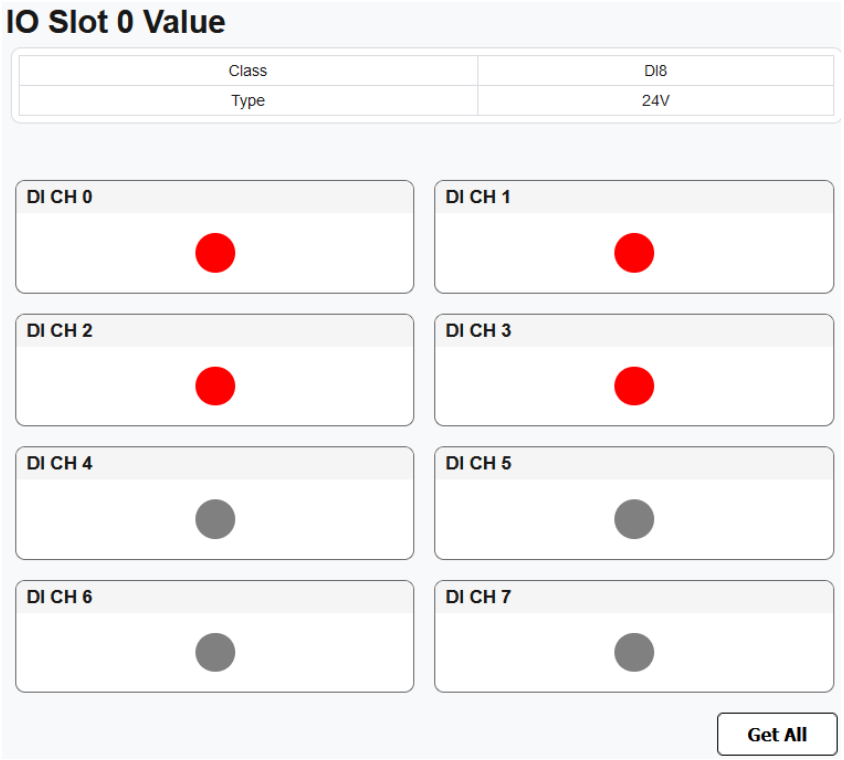



**Fig. 46 Digital Input Channel Value Access**

Fig. 46 shows the IO Slot 0 value page of the digital input channel numbers 0 to 7.

Only users with IO Value access rights are allowed to access this page. It is not visible for other users.

The IO values of all channels are read by clicking the *Get All* button.

#### 4.2.1.4.2 IO Channel Configuration

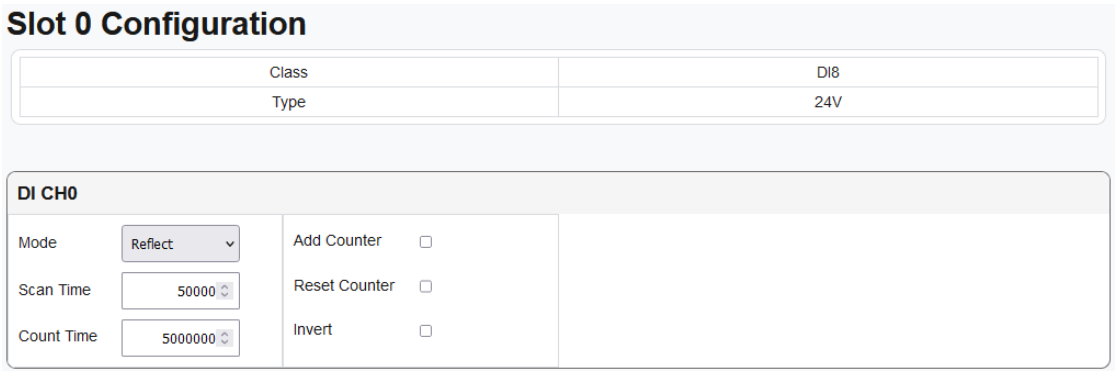



**Fig. 47 Digital Input Channel Configuration**

Fig. 47 shows the IO Slot 0 configuration page of digital input channels.

Only users with IO Config access rights are allowed to access this page. It is not visible for other users.

The IO Configuration parameters (→ 4.2.1.3) are updated and permanently saved by clicking the *Save* button.

#### 4.2.1.5 FRAME Interface

This section explains how the DI8 function class values and configuration parameters are accessed by using the FRAME interface (→ 5.4) and the LucidIoCtrl command line tool.

##### 4.2.1.5.1 IO Channel Value

The available IO value type (→ 5.2) depends on the operation mode of the IO channel.

| Mode | Supported FRAME Value Type | |
|---|---|---|
| Reflect | DI1 | Digital Logic Value 0 or 1 |
| Edge Detection | DI1 | Pending edge, logic value 0 or 1 |
| Count | CNT2 | Counter Value 0 … 65535 |

**Tab. 53 Digital Input IO Value Types**

###### 4.2.1.5.1.1 FRAME GetIo Command

The FRAME protocol command GetIo (→ 5.4.4.1) reads and returns the value of an IO channel number.

| **Command** | GetIo | **Access** | Read |
|---|---|---|---|
| **Opcode** | 0x46 | | |
| **LucidIoCtrl Arguments** | | | |
| **Call (-tL)** | `LucidIoCtrl -dtcp:[ip:port] -c[Channel] -tL -r` | | |
| **Return** | `CHn:v`<br>n: IO Channel Number<br>v: Digital Input Value | | |
| **Call (-tN)** | `LucidIoCtrl -dtcp:192.168.177.240:50001 -c[Channel] -tN -r` | | |
| **Return** | `CHn:v`<br>n: IO Channel Number<br>v: Count Value | | |

**Tab. 54 Digital Input FRAME GetIo Command and LucidIoCtrl Arguments**

LucidIoCtrl Command Line Tool Example

Read input channel 0. The module is operating in reflect or edge modes

```
      LucidIoCtrl -dtcp:192.168.177.240:50001 -c0 -tL -r
->    CH0:01
```

Input channel number 0 is "1".

Read input channel 0 in the case that the module is operating in count mode

```
      LucidIoCtrl –dtcp:192.168.177.240:50001 –c0 –tN –r
->    CH0:0x0064 (100)
```

The read counter value is 100

#### 4.2.1.5.1.2 FRAME GetIoGroup Command

The FRAME protocol command GetIoGroup (→ 5.4.4.2) reads and returns the values of a group of IO channel numbers.

| **Command** | GetIoGroup | **Access** | Read |
|---|---|---|---|
| **Opcode** | 0x48 | | |
| **LucidIoCtrl Command Line Tool** | | | |
| **Call (-tL)** | `LucidIoCtrl –dtcp:[ip:port] –c[Channels] –tL –r`<br><br>Channels:<br>Comma separated list of IO channel numbers e.g. -c0,1,3 | | |
| **Return** | List of values sorted from lower to higher IO channel numbers<br>`CHn:v`<br>n – IO Channel Number<br>v – Digital Input Value | | |
| **Call (-tN)** | `LucidIoCtrl –dtcp:[ip:port] –c[Channels] –tN –r`<br><br>Channels:<br>Comma separated list of IO channel numbers e.g. -c0,1,3 | | |
| **Return** | List of values sorted from lower to higher IO channel numbers<br>`CHn:v`<br>n – IO Channel Number<br>v – Counter Value | | |

**Tab. 55 Digital Input FRAME GetIoGroup command and LucidIoCtrl Arguments**

LucidIoCtrl Command Line Tool Example

Read input values of input channel numbers 0, 1 and 3:

```
      LucidIoCtrl –dtcp:192.168.177.240:50001 –c0,1,3 –tL –r
      CH0:00  CH1:01  CH3:01
```

### 4.2.1.5.2 IO Channel Configuration

IO channel configuration parameters are set by the FRAME protocol command SetParam (→ 5.4.4.5).

IO channel configuration parameters are read by the FRAME protocol command GetParam (→ 5.4.4.6).

#### 4.2.1.5.2.1 inDiValue

FRAME protocol gives access to *inDiValue* IO channel configuration parameter (→ 4.2.1.3.1).

| **Parameter** | *inDiValue* | **Address** | 0x1000 |
|---|---|---|---|
| | | **Parameter Type** | 1 Byte unsigned |
| **LucidIoCtrl Command Line Tool** | | | |
| **Parameter Name** | *inDiValue* | **Parameter Values** | 0x00 or 0x01 |
| **Call (Get)** | `LucidIoCtrl –dtcp:[ip:port] –c[Channel] –ginDiValue` | | |

**Tab. 56 IO Configuration Parameter inDiValue (FRAME Protocol and LucidIoCtrl)**

LucidIoCtrl Command Line Tool Example

Read inDiValue of input channel number 0:

```
      LucidIoCtrl –dtcp:192.168.177.240:50001 –c0 –ginDiValue
->    inDiValue=0
```

#### 4.2.1.5.2.2 inDiMode

FRAME protocol access to *inDiMode* IO configuration parameter (→ 4.2.1.3.2).

| **Parameter** | *inDiMode* | **Address** | 0x1100 |
|---|---|---|---|
| **Values** | Input Mode<br>**Mode** / **Mode Byte** / **Mode Value**<br>Inactive / 0x00 / inactive<br>Reflect / 0x01 / reflect<br>Rising edge / 0x10 / risingEdge<br>Falling edge / 0x11 / fallingEdge<br>Count / 0x20 / count | | |
| | | **Parameter Type** | 1 Byte unsigned |
| **LucidIoCtrl Command Line Tool** | | | |
| **Parameter Name** | *inDiMode* | **Parameter Values** | Mode Value |
| **Call (Set)** | `LucidIoCtrl –dtcp:[ip:port] –c[Channel] –sinDiMode=[Mode Value] {-p} {--default}` | | |
| **Call (Get)** | `LucidIoCtrl –dtcp:[ip:port] –c[Channel] –ginDiMode` | | |

**Tab. 57 IO Configuration Parameter inDiMode (FRAME Protocol and LucidIoCtrl)**

LucidIoCtrl Command Line Tool Example

Set operation mode of IO channel number 0 to count mode and make the setting persistent.

```
      LucidIoCtrl –dtcp:192.168.177.240:50001 –c0 –sinDiMode=count –p
```

Read the operation mode of IO channel number 0.

```
      LucidIoCtrl –dtcp:192.168.177.240:50001 –c0 –ginDiMode
->    inDiMode=count
```

#### 4.2.1.5.2.3 inDiFlags

FRAME protocol access to IO configuration parameter *inDiFlags* (→ 4.2.1.3.3).

The IO configuration parameter *inDiFlags* cannot be accessed by LucidIoCtrl directly. Use the bit parameters instead.

<table>
<tr><td>Parameter</td><td>inDiFlags</td><td>Address</td><td>0x1101</td></tr>
<tr><td>Values</td><td colspan="3">Consists of the following Bit Parameters
<table>
<tr><th>Bit Parameter</th><th>Bit Postion</th></tr>
<tr><td>inDiAddCounter</td><td>Bit 0</td></tr>
<tr><td>inDiResetCounterOnRead</td><td>Bit 1</td></tr>
<tr><td>inDiInverted</td><td>Bit 2</td></tr>
</table></td></tr>
<tr><td>Default Value</td><td>0x00</td><td>Parameter Type</td><td>1 Byte unsigned</td></tr>
</table>

**Tab. 58 IO Configuration Parameter inDiFlags (FRAME Protocol)**

When *inDiFlags* is changed by the SetParam FRAME command (→ 5.4.4.5), it must be done in a in a read-modify-write sequence in order to prevent overwriting other parameter bits.

#### 4.2.1.5.2.3.1 inDiInverted

FRAME protocol access to IO configuration parameter *inDiInverted* (→ 4.2.1.3.3.1).

| **Parameter** | *inDiInverted* | **Address** | 0x1101 (Bit 2) |
|---|---|---|---|
| **Values** | on/off | **Parameter Type** | 1 Bit |
| **LucidIoCtrl Command Line Tool** | | | |
| **Parameter Name** | *inDiInverted* | **Parameter Values** | on/off |
| **Call (Set)** | `LucidIoCtrl -dtcp:[ip:port] -c[Channel] -sinDiInverted=[Value] {-p} {--default}` | | |
| **Call (Get)** | `LucidIoCtrl -dtcp:[ip:port] -c[Channel] -ginDiInverted` | | |

**Tab. 59 IO Configuration Parameter Bit inDiInverted (FRAME Protocol and LucidIoCtrl)**

LucidIoCtrl Command Line Tool Example

Enable inversion of input signal of input channel number 0 and make the setting persistent.

```
      LucidIoCtrl -dtcp:192.168.177.240:50001 -c0 -sinDiInverted=on -p
```

Read inversion option of input channel number 0

```
      LucidIoCtrl -dtcp:192.168.177.240:50001 -c0 -ginDiInverted
->    inDiInverted=on
```

#### 4.2.1.5.2.3.2 inDiAddCounter

FRAME protocol access to IO configuration parameter inDiAddCounter (→ 4.2.1.3.3.2).

| **Parameter** | *inDiAddCounter* | **Address** | 0x1101 (Bit 0) |
|---|---|---|---|
| **Values** | on/off | **Parameter Type** | 1 Bit |
| **LucidIoCtrl Command Line Tool** | | | |
| **Parameter Name** | *inDiAddCounter* | **Parameter Values** | on/off |
| **Call (Set)** | `LucidIoCtrl -dtcp:[ip:port] -c[Channel] -sinDiAddCounter=[Value] {-p} {--default}` | | |
| **Call (Get)** | `LucidIoCtrl -dtcp:[ip:port] -c[Channel] -ginDiAddCounter` | | |

**Tab. 60 IO Configuration Parameter Bit inDiAddCounter (FRAME Protocol and LucidIoCtrl)**

LucidIoCtrl Command Line Tool Example

Enable add counter option for input channel number 0 and make the setting persistent.

```
LucidIoCtrl -dtcp:192.168.177.240:50001 -c0 -sinDiAddCounter=on -p
```

Read add counter option of input channel number 0

```
LucidIoCtrl -dtcp:192.168.177.240:50001 -c0 -ginDiAddCounter
->  inDiAddCounter=on
```

#### 4.2.1.5.2.3.3 inDiResetCounterOnRead

FRAME protocol access to IO configuration parameter *inDiResetCounterOnRead* (→ 4.2.1.3.3.3).

| **Parameter** | *inDiResetCounterOnRead* | **Address** | 0x1101 (Bit 1) |
|---|---|---|---|
| **Values** | on/off | **Parameter Type** | 1 Bit |
| **LucidIoCtrl Command Line Tool** | | | |
| **Parameter Name** | *inDiResetCounterOnRead* | **Parameter Values** | on/off |
| **Call (Set)** | `LucidIoCtrl -dtcp:[ip:port] -c[Channel] -sinDiResetCounterOnRead=[Value] {-p} {--default}` | | |
| **Call (Get)** | `LucidIoCtrl -dtcp:[ip:port] -c[Channel] -ginDiResetCounterOnRead` | | |

**Tab. 61 Configuration Parameter Bit inDiResetCounterOnRead (FRAME Protocol and LucidIoCtrl)**

#### 4.2.1.5.2.4 inDiScanTime

FRAME protocol access to IO configuration parameter *inDiScanTime* (→ 4.2.1.3.4).

| **Parameter** | *inDiScanTime* | **Address** | 0x1111 |
|---|---|---|---|
| | | **Parameter Type** | 4 Bytes unsigned |
| **LucidIoCtrl Command Line Tool** | | | |
| **Parameter Name** | *inDiScanTime* | **Parameter Values** | Time [µs] |
| **Call (Set)** | `LucidIoCtrl -dtcp:[ip:port] -c[Channel] -sinDiScanTime=[Time]{-p} {--default}` | | |
| **Call (Get)** | `LucidIoCtrl -dtcp:[ip:port] -c[Channel] -ginDiScanTime` | | |

**Tab. 62 IO Configuration Parameter inDiScanTime (FRAME Protocol and LucidIoCtrl)**

LucidIoCtrl Command Line Tool Example

Set $T_{Scan}$ of input channel number 0 to 1.5 s and make the setting persistent.

```
LucidIoCtrl -dtcp:192.168.177.240:50001 -c0 -sinDiScanTime=1500000 -p
```

Read $T_{Scan}$ parameter of input channel number 0

```
      LucidIoCtrl -dtcp:192.168.177.240:50001 -c0 -ginDiScanTime4
->    inDiScanTime=1500000
```

#### 4.2.1.5.2.5 inDiCountTime

FRAME protocol access to IO configuration parameter *inDiCountTime* (→ 4.2.1.3.5).

| **Parameter** | *inDiCountTime* | **Address** | 0x1112 |
|---|---|---|---|
| | | **Parameter Type** | 4 Bytes unsigned |
| **LucidIoCtrl Command Line Tool** | | | |
| **Parameter Name** | *inDiCountTime* | **Parameter Values** | Time [µs] |
| **Call (Set)** | `LucidIoCtrl -dtcp:[ip:port] -c[Channel] -sinDiCountTime=[Time] {-p} {--default}` | | |
| **Call (Get)** | `LucidIoCtrl -dtcp:[ip:port] -c[Channel] -ginDiCountTime` | | |

**Tab. 63 IO Configuration Parameter inDiCountTime (FRAME Protocol and LucidIoCtrl)**

LucidIoCtrl Command Line Tool Example

Set $T_{Count}$ of input channel number 0 to 10 s and make the setting persistent.

```
      LucidIoCtrl -dtcp:192.168.177.240:50001 -c0 -sinDiCountTime=10000000 -p
```

Read $T_{Count}$ parameter of input channel number 0

```
      LucidIoCtrl -dtcp:192.168.177.240:50001 -c0 -ginDiCountTime
      ->inDiCountTime=10000000
```

### 4.2.1.6 JSON Interface

This section explains how to access DI8 IO channel value and configuration using the JSON interface (→ 3.3.14).

#### 4.2.1.6.1 IO Channel Value

##### 4.2.1.6.1.1 Read IO Channel Value

JSON protocol reads the IO channel values by the command IoGet (→ 5.3.2.4.1).

Example:

Read the input channels 0, 1, 8, and 15 of a LIOT16-DI8-24V-AI8-20MA module with 8 digital 24V inputs and 8 analog 0-20mA inputs.

Request:

```
      {"Cmd":"IoGet",
       "SessionId: 123456789,
       "Values":[{"Id":0, "Type":0},
                 {"Id":1, "Type":0},
                 {"Id":8, "Type":35},
                 {"Id":15, Type":29}]}
```

Response:

```
      {"Status":"OK",
       "Values":[{"Id":0, "Type":0, "Value":true},
                 {"Id":1, "Type":0, "Value":true},
                 {"Id":8, Type":35, "Value":15000000}
```

```
{"Id":15, Type":29, "Value":5000000}]}
```

The response returns for the two digital input channels IO value of "1", 15mA for the analog input channel 8 and 5V for analog input channel 15.

#### 4.2.1.6.2 IO Channel Configuration

JSON protocol accesses the IO channel configuration parameters by commands IoCfgSet (→ 5.3.2.4.3) and IoCfgGet (→ 5.3.2.4.3).

| | |
|---|---|
| "DI":{<br>"Mode": *modeByte*,<br>"AddCnt": *addCounterFlag*,<br>"RstCntRd": *resetCounterOnReadFlag*,<br>"Invert": *invertFlag*,<br>"ScanTime": *scanTime*,<br>"CntTime": *countTime*} | |

**Tab. 64 Digital Input Channel Configuration JSON Object**

| Name | Type | Description |
|---|---|---|
| Mode | Number | IO Configuration Parameter *inDiMode* (→ 4.2.1.3.2)<br>Parameter value Mod Byte in Tab. 57. |
| AddCnt | Boolean | IO Configuration Parameter *inDiAddCounter* (→ 4.2.1.3.3.2) |
| RstCntRd | Boolean | IO Configuration Parameter *inDiResetCounterOnRead* (→ 4.2.1.3.3.3) |
| Invert | Boolean | IO Configuration Parameter *inDiInverted* (→ 4.2.1.3.3.1) |
| ScanTime | Number | IO Configuration Parameter *inDiScanTime* (→ 4.2.1.3.4)<br>Parameter value *scanTime* in µs. |
| CntTime | Number | IO Configuration Parameter *inDiCountTime* (→ 4.2.1.3.5)<br>Parameter value *countTime* in µs. |

**Tab. 65 Digital Input Channel Configuration JSON Object Values**

Fields of the digital input channel configuration JSON object are optional.

##### 4.2.1.6.2.1 Read Configuration

The JSON command IoCfgGet (→ 5.3.2.4.3) reads the IO channel configuration parameters of an IO channel.

Example

The device LIOT-DI8-24V-DI8-10V has 16 digital input channels.
The configuration of the channel numbers 0 and 15 are read:

Request

```
{"Cmd": "IoCfgGet",
 "SessionId": 123456789,
 "ValueIds":[0,15]}
```

Response

```
{"Status": "OK",
 "Configs":[
  "Id": 0,
  "Type": 0,
  "DI":{
   "Mode": 32,
   "AddCnt": false,
   "RstCntRd": false,
   "Invert": true,
   "ScanTime": 50000,
   "CountTime": 2000000}},{
  "Id": 15,
  "Type": 0,
  "DI":{
   "Mode": 32,
   "AddCnt": false,
   "RstCntRd": false,
   "Invert": true,
   "ScanTime": 50000,
   "CountTime": 2000000}}]}
```

The response returns the IO configuration parameters of channel numbers 0 and 15. They operate in count mode, inputs are inverted, scan time is 50ms and count time 2s.

#### 4.2.1.6.2.2 Update Configuration

The JSON command IoCfgSet (→ 5.3.2.4.3) writes the IO channel configuration parameters of an IO channel.

If an IO configuration parameter is not included in the object, it remains unchanged.

Example

The device LIOT-DI8-24V-DI8-10V has 16 digital input channels.
The configuration of the channel numbers 0 and 15 should be updated:

> Parameters *inDiCountTime* should be set to 2s, *inDiInverted* should be activated, channels should operate in count mode.
> All other parameters should remain unchanged.

Request

```
{"Cmd": "IoCfgSet",
 "SessionId": 123456789,
 "Configs":[{
  "Id": 0,
  "Type": 0,
  "DI":{
   "Mode": 32,
   "Invert": true,
   "ScanTime": 2000000}},{
  "Id": 15,
  "Type": 0,
  "DI":{
```

```
"Mode": 32,
"Invert": true,
"CntTime": 2000000}}]}
```

#### 4.2.1.7 MQTT Client

The digital input channel IO values are accessible by the MQTT client (→ 3.3.10).

The IO channel values are selected by value identifiers (→ 3.3.10.4.1). They are formatted according to the value format settings (Tab. 66).

| Mode | Value Formats |
|---|---|
| Reflect | Digital 1 / 0 (Auto Format)<br>Digital On / Off<br>Digital true / false |
| Edge Detection | Same as for reflect mode |
| Count | Numeric (Auto Format) |

**Tab. 66 Digital Input Channel Value Formats**

#### 4.2.1.8 Modbus/TCP Server

The IO channel values and configuration parameters are accessible by Modbus/TCP registers (→ 3.3.11.2).

IO channel values are accessed by holding registers. Register write accesses are successful, but ignored for digital input channels.

#### 4.2.1.9 Data Logging

The IO channel values can be logged (→ 3.4.3).

The IO channel values are selected by value identifiers (→ 3.3.10.4.1). They are formatted according to the value format settings (→ 3.3.10.4.4, Tab. 66).

## 4.2.2 Digital Output (DO8 Function Class)

The DO8 function class controls 8 digital output lines.

| Function Class | Value | Channels |
|---|---|---|
| DO8 | 0x1010 | 8 |

**Tab. 67 Digital Output Function Classes**

| Function Class Type | Value | Output Type |
|---|---|---|
| I | 0x1000 | Solid State Relay (SSR) |
| 0 | 0x1200 | Open Collector |

**Tab. 68 Digital Output Function Class Types**

Tab. 67 and Tab. 68 list the DO8 function class and it's supported function class types.

### 4.2.2.1 IO Signal

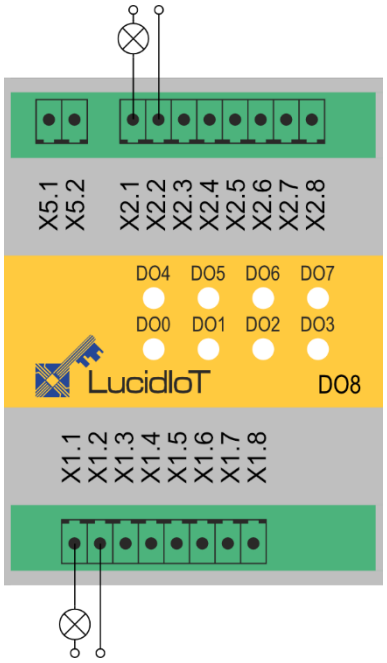



**Fig. 48 DO8-I at IO Slot 0**

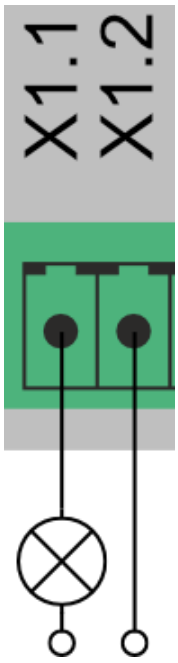



**Fig. 49 DO8-I Signal**

Fig. 48 shows the connection section of the DO8 function installed at IO Slot 0.

A power load (e.g. a lamp) is connected to X1.1 and X1.2 (DO0) and X2.1 and X2.2 (DO4).

Fig. 49 shows the lamp connected to the IO terminals X1.1 and X1.2 (DO0) in detail.

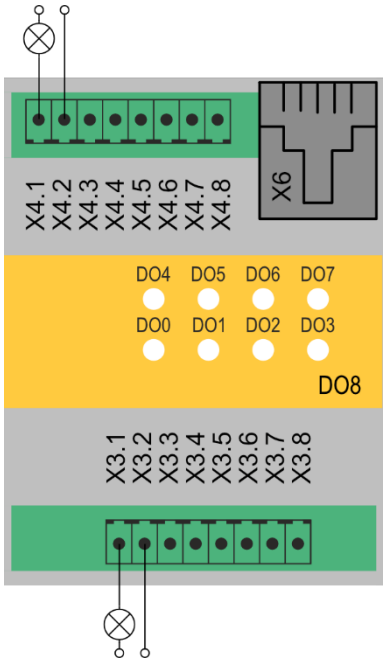



**Fig. 50 DO8 at IO Slot 1**

Fig. 50 shows the IO terminals of the DO8 function class installed at IO Slot 1.

Two power loads (e.g. lamps) are connected to X3.1 and X3.2 (DO0) and X4.1 and X4.2 (DO4).

#### 4.2.2.1.1 Solid State Relay (-I Type)

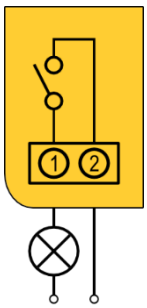



**Fig. 51 SSR Output**

Fig. 51 shows the principle of a digital output channel with solid state relays (SSR).

When the digital output channel is set to high, the SSR connects IO terminal 1 (e.g. X1.1) with IO terminal 2 (e.g. X1.2), closing the circuit.

The polarity of the signal is not relevant for the SSR output channels

The SSR output channels are opto-isolated, protecting the electronic behind the SSR. The SSR output channels do not share any contacts and are independent.

The IO terminals of the DO8-I function class and their IO channel numbers are listed in Tab. 69. The state of the output is indicated by a red LED on the front panel.

| IO Terminal | IO Slot | Signal | IO Channel Number |
|---|---|---|---|
| X1.1 | 0 | DO0 | 0 |
| X1.2 | | | |
| X1.3 | | DO1 | 1 |
| X1.4 | | | |
| X1.5 | | DO2 | 2 |
| X1.6 | | | |
| X1.7 | | DO3 | 3 |
| X1.8 | | | |
| X2 | | DO4 … DO7 | 4 – 7 |
| X3 | 1 | DO0 … DO3 | 8 – 11 |
| X4 | | DO4 … DO7 | 12 – 15 |

**Tab. 69 DO8-I IO Terminal Connector**

The SSR outputs are not protected against overcurrent and overvoltage. $U_{SSRMax}$ and $I_{SSRMax}$ limits must be considered. Otherwise, the output may be damaged.

If inductive loads are controlled, additional protection may be necessary in order to protect the SSR from excessive high voltage.

SSR outputs support reflect mode, duty-cycle mode and on-off mode.
For duty-cycle and on-off modes the minimum on and off times are limited to $T_{SSRMin}$.

#### 4.2.2.1.2 Open Collector Transistor Outputs (-O Type)

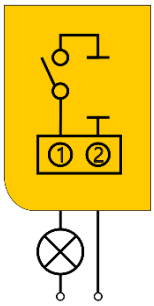



**Fig. 52 Open Collector**

Fig. 52 shows the principle of the open collector digital output channel.

When the digital output channel is set to high, the transistor connects IO terminal 1 (e.g. X1.1) with ground, closing the circuit.

IO terminals with uneven numbers (e.g. X1.1 or X4.1) are connected to positive potential.
IO terminals with even numbers (e.g. X1.2 or X4.2) are internally connected to a common ground signal.

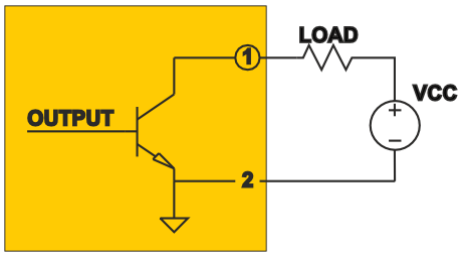



**Fig. 53 Open Collector Circuit**

The internal circuit of the open collector output is shown in Fig. 53. When the OUTPUT control signal is high, the transistor closes the circuit. This means that IO terminal 1 is connected to ground and the externally applied voltage source VCC powers the load.

The IO terminals of the DO8-O function class are listed in Tab. 70.
The state of the output is indicated by a red LED on the front panel.

| IO Terminal | IO Slot | Signal | IO Channel Number |
|---|---|---|---|
| X1.1 | 0 | DO0 + | 0 |
| X1.2 | | GND | |
| X1.3 | | DO1 + | 1 |
| X1.4 | | GND | |
| X1.5 | | DO2 + | 2 |
| X1.6 | | GND | |
| X1.7 | | DO3 + | 3 |
| X1.8 | | GND | |
| X2 | | DO4 … DO7 | 4 – 7 |
| X3 | 1 | DO0 … DO3 | 8 – 11 |
| X4 | | DO4 … DO7 | 12 – 15 |

**Tab. 70 DO8-O IO Terminal Connector**

The open collector outputs are not protected against overcurrent and overvoltage. $I_{OCMax}$ and $U_{OCMax}$ limits must be considered in. Otherwise, the device may be damaged.

The open collector outputs are not protected against wrong polarization of the applied voltages.

Open collector outputs support reflect mode, duty-cycle mode and on-off mode.

For duty-cycle and on-off modes the minimum on and off times are limited to $T_{OCMin}$.

#### 4.2.2.1.3 Inverted IO Value

If the IO configuration parameter *outDiInverted* is set to "on", the output signal is inverted.

### 4.2.2.2 Operation Modes

Each digital output channel can work in one of the following modes:

- Reflect mode
- Duty-cycle mode
- On-off mode

Each digital output channel operates according to its IO configuration parameters (→ 4.2.2.2.1.4)

If not otherwise stated, IO configuration parameters are updated when the IO value is set.

#### 4.2.2.2.1.1 Reflect Mode

In reflect mode, the digital output channel is directly controlled by the IO channel value.

Writing "1" to the IO channel value sets the output to "on" state, writing "0" to the IO channel value clears the to "off" state.

Reading the IO channel value returns the current state of the IO channel.

The timing of the signals is limited and fast switching (less of 50 ms) would be inaccurate because of processing and communication latency. In applications needing deterministic processing, duty-cycle or on-off mode should be considered.

#### 4.2.2.2.1.2 Duty-Cycle Mode (PWM)

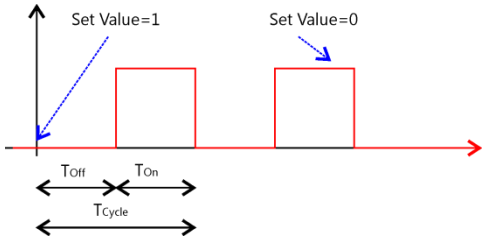



**Fig. 54 Digital Output in Duty-Cycle Mode**

In duty-cycle mode the digital output state is periodically switched on and off (PWM, pulse-width-modulation).

Fig. 54 shows a periodical signal generated by duty-cycle mode.

Setting the IO channel value of the output channel to "1" starts processing.
Setting the IO channel value of the output channel to "0" stops processing.

Reading the IO channel value returns the state of processing.

The timing of the signal is defined by IO configuration parameters:

- The parameter $T_{Cycle}$ defines the cycle time (period).
  It is set by the IO configuration parameter *outDiCycleTime*.
- The IO configuration parameter *outDiDutyCycle* sets the relation between on-time $T_{On}$ and off-time $T_{Off}$.

The on-phase time equals to $T_{On} = \frac{T_{Cycle}}{1000} * outDiDutyCycle$

The off-phase time equals to $T_{Off} = T_{Cycle} - T_{On}$

If the values of $T_{On}$ or $T_{Off}$ violate the limits of $T_{OutDiMin}$, on-phase or off-phase are skipped.

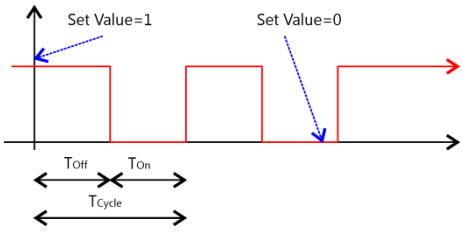



**Fig. 55 Digital Output in Duty-Cycle Mode (inverted)**

If the parameter *outDiInverted* set to "on", the state of the output switch is inverted (Fig. 55). The output is switched "on" during $T_{Off}$ (off-phase) and "off" during $T_{On}$ (on-phase).

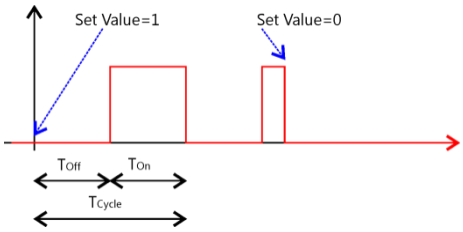



**Fig. 56 Digital Output in Duty-Cycle Mode (Cancel)**

If the IO value is set to "0" during on-phase, the behavior depends on IO channel configuration parameter *outDiCanCancel*.

If the parameter *outDiCanCancel* is set to "off", processing stops after the on-phase ended. This behavior is shown in Fig. 54.

If the parameter *outDiCanCancel*. is set to "on", processing stops immediately (Fig. 56).

Updating of Parameters

If processing is running, updates of the IO Configuration Parameters *outDiCycleTime* and *outDiDutyCycle* are applied immediately.

LucidIoCtrl Command Line Tool Example

Configure digital output channel number 0 for Duty-Cycle mode

```
LucidIoCtrl -dtcp:192.168.177.240:50001 -c0 -soutDiMode=dutyCycle
```

Start processing of PWM signal for digital output channel number 0

```
LucidIoCtrl -dtcp:192.168.177.240:50001 -c0 -tT -w1
```

The digital output channel generates the signal according to configured *outDiCycleTime* and *outDiDutyCycle*.

Changing $T_{Cycle}$ to 2 s

```
LucidIoCtrl -dtcp:192.168.177.240:50001 -c0 -soutDiCycleTime=2000000
```

Set DutyCycle = 75%

```
LucidIoCtrl -dtcp:192.168.177.240:50001 -c0 -soutDiDutyCycle=750
```

The digital output channel is switched 1.5 s to on-phase, 0.5 s to off-phase.

Disable digital output channel number 0.

```
LucidIoCtrl -dtcp:192.168.177.240:50001 -c0 -tT -w0
```

#### 4.2.2.2.1.3 On / Off Mode

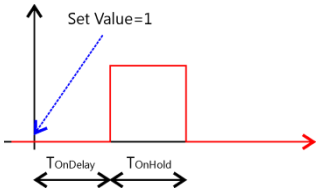



**Fig. 57 Digital Output in On-Off Mode**

In on-off mode, the digital output generates a one-time sequence pattern shown in Fig. 57.

Writing "1" to the IO value of the digital output channel starts processing of the output signal and starts the $T_{OnDelay}$ interval (off-phase). After $T_{OnDelay}$ has passed, it changes to on-phase and $T_{OnHold}$ interval starts. After $T_{OnHold}$ time has passed, the output channel finally changes back to initial off-phase and the sequence finishes.

Writing "0" to the IO channel value during off-phase stops processing, preventing the output entering on-phase.

Reading the IO channel value returns "1" if the processing is running, "0" if not started or ended.

The timing of the signal is defined by IO configuration parameters:

- The parameter $T_{OnDelay}$ defines durance of off-phase.
  It is set by the IO configuration parameter *outDiOnDelay*.
- The parameter $T_{OnHold}$ defines durance of on-phase.
  It is set by the IO configuration parameter *outDiOnHold*.

If the values of $T_{OnHold}$ or $T_{OnDelay}$ are outside the limits of $T_{OutDiMin}$, off-phase or on-phase are skipped.

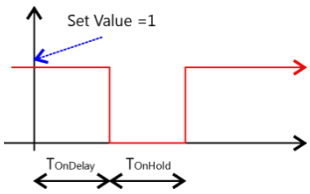



**Fig. 58 Digital Output in On-Off Mode (inverted)**

If the parameter *outDiInverted* set to "on", the state of the output switch is inverted (Fig. 58). The output is switched "on" during $T_{OnDelay}$ (off-phase) and "off" during $T_{OnHold}$ (on-phase).

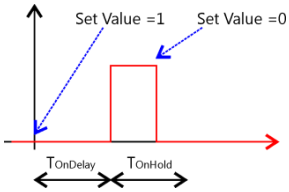



**Fig. 59 Digital Output in On-Off Mode (Cancel)**

If the IO value is set to "0" during on-phase, the behavior depends on IO channel configuration parameter *outDiCanCancel*.

Ignore, if the parameter *outDiCanCancel* is set to "off".

If parameter *outDiCanCancel*. is set to "on", processing stops immediately, returning to off-phase (Fig. 59).

If the IO value is set to "0" during on-phase, the behavior depends on IO channel configuration parameter *outDiCanCancel*.

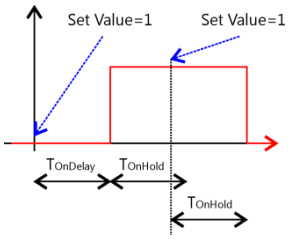



**Fig. 60 Digital Output in On-Off Mode (Retrigger)**

If a "1" is written to the IO value during on-phase, the behavior depends on IO channel configuration parameter *outDiCanRetrigger*.

Ignore, if parameter *outDiCanRetrigger is set to "off*".

*If parameter outDiCanRetrigger is set to "on", retriggers the on-phase by restarting TOnHold time.*

LucidIoCtrl Command Line Tool Example

Configure digital output channel number 0 for on-off mode

```
LucidIoCtrl -dtcp:192.168.177.240:50001 -c0 -soutDiMode=onoff
```

Assume that IO configuration parameters defaults. $T_{OnDelay}$ = 1 s and $T_{OnHold}$ = 1

Writing IO value "1" sets digital output channel numer 0 after 1 s to "on" for 1 s.

```
LucidIoCtrl -dtcp:192.168.177.240:50001 -c0 -tL -w1
```

#### 4.2.2.2.1.4 Inactive Mode

In inactive operation mode, the digital output is switched off permanently.

### 4.2.2.3 Configuration Parameters

#### 4.2.2.3.1 outDiValue

The IO configuration parameter *outDiValue* contains the IO channel value.

The IO channel value is initialized with IO configuration parameter *outDiValue* on system start.

| **Parameter** | *outDiValue* | **Access** | Read / Write |
|---|---|---|---|
| **Values** | IO Value | | |
| **Default Value** | 0x00 | **Parameter Type** | - |

**Tab. 71 IO Channel Configuration Parameter outDiValue**

#### 4.2.2.3.2 outDiMode

The IO configuration parameter *outDiMode* specifies the operation mode (→ 4.2.2.2) of the digital output channel.

If *outDiMode* is changed while IO value is set, the IO value is cleared.
The updated operation mode starts by setting IO value again.

| **Parameter** | *outDiMode* | **Access** | Read / Write |
|---|---|---|---|
| **Values** | Operation Mode<br>inactive, reflect, on-off, duty-cycle | | |
| **Default Value** | reflect | **Parameter Type** | - |

**Tab. 72 IO Channel Configuration Parameter outDiMode**

#### 4.2.2.3.3 outDiFlags

The IO configuration parameter *outDiFlags* specifies the operation flags of the digital output channel.

##### 4.2.2.3.3.1 outDiInverted

The IO configuration parameter bit *outDiInverted* specifies if the IO channel value is inverted. (→ 4.2.2.1.3)

| **Parameter** | *outDiInverted* | **Access** | Read / Write |
|---|---|---|---|
| **Values** | on/off | | |
| **Default Value** | off | **Parameter Type** | 1 Bit |

**Tab. 73 IO Channel Configuration Parameter Bit outDiInverted**

##### 4.2.2.3.3.2 outDiCanCancel

The IO configuration parameter bit *outDiCanCancel* specifies if the output processing can be canceled in duty-cycle and on-off modes.

| **Parameter** | *outDiCanCancel* | **Access** | Read / Write |
|---|---|---|---|
| **Values** | on/off | | |
| **Default Value** | off | **Parameter Type** | 1 Bit |

**Tab. 74 IO Channel Configuration Parameter Bit outDiCanCancel**

##### 4.2.2.3.3.3 outDiCanRetrigger

The IO configuration parameter bit *outDiCanRetrigger* specifies if the on-phase can be retriggered in on-off mode.

| **Parameter** | *outDiCanRetrigger* | **Access** | Read / Write |
|---|---|---|---|
| **Values** | on/off | | |
| **Default Value** | off | **Parameter Type** | 1 Bit |

**Tab. 75 IO Channel Configuration Parameter Bit outDiCanRetrigger**

#### 4.2.2.3.4 outDiCycleTime

The IO configuration parameter *outDiCycleTime* specifies the $T_{Cycle}$ time when operating in duty-cycle mode.

If the output channel processing is running, changes to IO configuration parameter *outDiCycleTime* are applied immediately.

| **Parameter** | *outDiCycleTime* | **Access** | Read / Write |
|---|---|---|---|
| **Values** | $T_{Cycle}$ in µs (microseconds)<br>$T_{OutDiMin} \leq T_{Cycle} \leq 1$ h | | |
| **Default Value** | 1,000,000 (1 s) | **Parameter Type** | 4 Bytes unsigned |

**Tab. 76 IO Channel Configuration Parameter outDiCycleTime**

#### 4.2.2.3.5 outDiDutyCycle

The IO configuration parameter *outDiDutyCycle* specifies the duty-cycle when operating in duty-cycle mode.

If the output channel processing is running, changes to IO configuration parameter *outDiDutyCycle* are applied immediately.

| **Parameter** | *outDiDutyCycle* | **Access** | Read / Write |
|---|---|---|---|
| **Values** | Duty-cycle in ‰ (1 / 1000) | | |
| **Default Value** | 500 (50%) | **Parameter Type** | 2 Bytes unsigned |

**Tab. 77 IO Channel Configuration Parameter outDiDutyCycle**

#### 4.2.2.3.6 outDiOnDelay

The IO configuration parameter *outDiOnDelay* specifies the $T_{OnDelay}$ time in on-off mode.

| **Parameter** | *outDiOnDelay* | **Access** | Read / Write |
|---|---|---|---|
| **Values** | $T_{OnDelay}$ in µs (microseconds),<br>$T_{OutDiMin} \leq T_{OnDelay} \leq 1$ h | | |
| **Default Value** | 1,000,000 (1 s) | **Parameter Type** | 4 Bytes unsigned |

**Tab. 78 IO Channel Configuration Parameter outDiOnDelay**

#### 4.2.2.3.7 outDiOnHold

The IO configuration parameter *outDiOnHold* specifies the $T_{OnHold}$ time in on-off mode.

| **Parameter** | *outDiOnHold* | **Access** | Read / Write |
|---|---|---|---|
| **Values** | $T_{OnHold}$ in µs (microseconds)<br>$T_{OutDiMin} \leq T_{OnHold} \leq 1$ h | | |
| **Default Value** | 1,000,000 (1 s) | **Parameter Type** | 4 Bytes unsigned |

**Tab. 79 IO Channel Configuration Parameter outDiOnHold**

### 4.2.2.4 Web Interface

This section explains how the DO8 IO channel value and configuration are accessed using the web interface.

#### 4.2.2.4.1 IO Channel Value

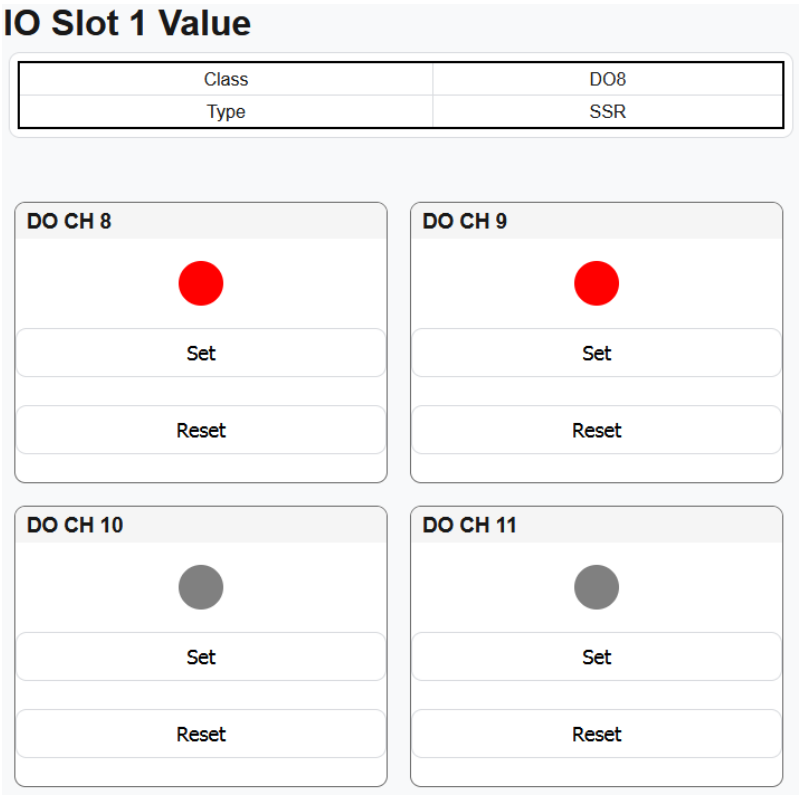



**Fig. 61 Digital Output Channel Value Access**

Fig. 61 shows the IO Slot 1 value page of the digital output channel.

Only users with IO Value access rights are allowed to access this page. It is not visible for other users.

A digital output channel can be set by clicking the *Set* button, and reset by clicking the *Reset* button.

All digital output channels can be set by clicking the *Set All* button, and reset by clicking the *Reset All* button.

#### 4.2.2.4.2 IO Channel Configuration

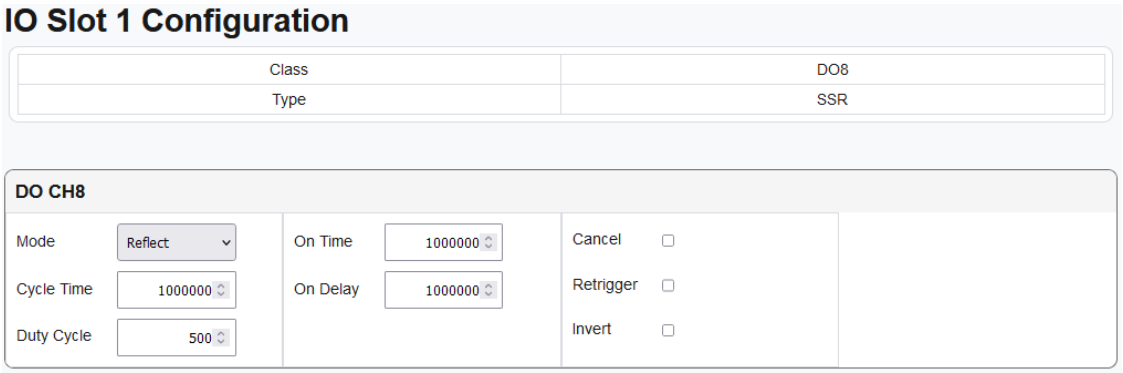



**Fig. 62 Digital Output Channel Configuration**

Fig. 62 shows the IO Slot 1 configuration page of digital output channels.

Only users with IO Config access rights are allowed to access this page. It is not visible for other users.

The IO Configuration parameters (→ 4.2.2.3) are updated and permanently saved by clicking the *Save* button.

### 4.2.2.5 FRAME Interface

This section explains how the DO8 function class values and configuration parameters are accessed by using the FRAME interface (→ 5.4) and the LucidIoCtrl command line tool.

#### 4.2.2.5.1 IO Channel Value

Logic IO value type (→ 5.2) is used in all operation modes.

| Mode | Suported Frame Value Type |
|---|---|
| reflect, duty-cycle, on-off | DI1 |

**Tab. 80 Digital Output IO Value Types**

##### 4.2.2.5.1.1 FRAME GetIo Command

→ 4.2.1.5.1.1

##### 4.2.2.5.1.2 FRAME GetIoGroup Command

→ 4.2.1.5.1.2

##### 4.2.2.5.1.3 FRAME SetIo Command

The FRAME protocol command SetIo (→ 5.4.4.3) writes the value of an IO channel number.

| **Command** | SetIo | **Access** | Write |
|---|---|---|---|
| **Opcode** | 0x40 | **Values** | 0/1 |
| **LucidIoControl Command Line Tool** | | | |
| **Call (-tL)** | `LucidIoCtrl –dtcp:[ip:port] –c[Channel] –tL –w[Value]` | | |

**Tab. 81 Digital Output FRAME SetIo Command and LucidIoCtrl Arguments**

LucidIoCtrl Command Line Tool Example

Set digital output channel number 0 to "1":

```
LucidIoCtrl –dtcp:192.168.177.240:50001 –c0 –tL –w1
```

#### 4.2.2.5.1.4 FRAME SetIoGroup Command

The FRAME protocol command SetIoGroup (→ 5.4.4.4) writes the values of a group of IO channel numbers.

| **Command** | SetIoGroup | **Access** | Write |
|---|---|---|---|
| **Opcode** | 0x42 | | |
| **LucidIoCtrl Command Line Tool** | | | |
| **Call (-tL)** | `LucidIoCtrl –dtcp:[ip:port] –c[Channels] –tL –w[Values]`<br><br>Channels:<br>Comma separated list of channels e.g. –c0,1,3<br><br>Values:<br>Comma separated list of values to set e.g. -w1,1,0 | | |

**Tab. 82 Digital Output FRAME SetIoGroup Command and LucidIoCtrl Arguments**

LucidIoCtrl Command Line Tool Example

Set output channel numbers 0 to "1", 2 to "1" and 3 to "0":

```
LucidIoCtrl –dtcp:192.168.177.240:50001 –c0,2,3 –tL –w1,1,0
```

### 4.2.2.5.2 IO Channel Configuration

IO channel configuration parameters are set by the FRAME protocol command SetParam (→ 5.4.4.5).

IO channel configuration parameters are read by the FRAME protocol command GetParam (→ 5.4.4.6).

#### 4.2.2.5.2.1 outDiValue

FRAME protocol gives access to *outDiValue* IO channel configuration parameter (→ 4.2.2.3.1).

| **Parameter** | *outDiValue* | **Address** | 0x1000 |
|---|---|---|---|
| | | **Parameter Type** | 1 Byte unsigned |
| **LucidIoCtrl Command Line Tool** | | | |
| **Parameter Name** | *outDiValue* | **Parameter Values** | 0x00 or 0x01 |
| **Call (Set)** | `LucidIoCtrl -dtcp:[ip:port] -c[Channel] -soutDiValue=[Value] {-p} {--default}` | | |
| **Call (Get)** | `LucidIoCtrl -dtcp:[ip:port] -c[Channel] -goutDiValue` | | |

**Tab. 83 IO Configuration Parameter outDiValue (FRAME Protocol and LucidIoCtrl)**

LucidIoCtrl Command Line Tool Example

Set outDiValue of output channel number 0 to "1" and make the setting persistent:

```
LucidIoCtrl -dtcp:192.168.177.240:50001 -c0 -soutDiValue=1 -p
```

Read outDiValue or state of output channel 0:

```
      LucidIoCtrl -dtcp:192.168.177.240:50001 -c0 -goutDiValue
->    outDiValue=1
```

#### 4.2.2.5.2.2 outDiMode

FRAME protocol gives access to *outDiMode* IO configuration parameter (→ 4.2.2.3.2).

| **Parameter** | *outDiMode* | **Access** | Read / Write |
|---|---|---|---|
| **Address** | 0x1100 | | |
| **Values** | Output Mode<br>**Mode** / **Mode Byte** / **Mode Value**<br>Inactive / 0x00 / inactive<br>Reflect / 0x01 / reflect<br>On-Off / 0x08 / onoff<br>Duty-Cycle / 0x0A / dutyCycle | | |
| | | **Parameter Type** | 1 Byte unsigned |
| **LucidIoCtrl Command Line Tool** | | | |
| **Parameter Name** | *outDiMode* | **Parameter Values** | Mode Value |
| **Call (Set)** | `LucidIoCtrl -dtcp:[ip:port] -c[Channel] -soutDiMode=[Mode Value] {-p} {--default}` | | |
| **Call (Get)** | `LucidIoCtrl -dtcp:[ip:port] -c[Channel] -goutDiMode` | | |

**Tab. 84 IO Configuration Parameter outDiMode (FRAME Protocol and LucidIoCtrl)**

LucidIoCtrl Command Line Tool Example

Set operation mode of IO channel number 0 to duty-cycle mode and make the setting persistent.

```
LucidIoCtrl -dtcp:192.168.177.240:50001 -c0 -soutDiMode=dutyCycle -p
```

Read the operation mode of IO channel number 0.

```
      LucidIoCtrl -dtcp:192.168.177.240:50001 -c0 -goutDiMode
->    outDiMode=dutyCycle
```

#### 4.2.2.5.2.3 outDiFlags

FRAME protocol gives access to IO configuration parameter *outDiFlags* (→ 4.2.2.3.3)

The parameter *outDiFlags* cannot be accessed by LucidIoCtrl directly. Use the bit parameters instead.

| **Parameter** | *outDiFlags* | **Address** | 0x1101 |
|---|---|---|---|
| **Values** | Consists of the following Bit Parameters<br>**Bit Parameter** / **Bit Postion**<br>*outDiCanRetrigger* / Bit 0<br>*outDiCanCancel* / Bit 1<br>*outDiInverted* / Bit 2 | | |

**Tab. 85 IO Configuration Parameter outDiFlags (FRAME Protocol and LucidIoCtrl)**

##### 4.2.2.5.2.3.1 outDiInverted

FRAME protocol access to IO configuration parameter *outDiInverted*. (→ 4.2.2.3.3.1).

| **Parameter** | *outDiInverted* | **Address** | 0x1101 (Bit 2) |
|---|---|---|---|
| **Values** | on/off | **Parameter Type** | 1 Bit |
| **LucidIoCtrl Command Line Tool** | | | |
| **Parameter Name** | *outDiInverted* | **Parameter Values** | on/off |
| **Call (Set)** | `LucidIoCtrl -dtcp:[ip:port] -c[Channel] -soutDiInverted=[Value] {-p} {--default}` | | |
| **Call (Get)** | `LucidIoCtrl -dtcp:[ip:port] -c[Channel] -goutDiInverted` | | |

**Tab. 86 IO Configuration Parameter Bit outDiInverted (FRAME Protocol and LucidIoCtrl)**

LucidIoCtrl Command Line Tool Example

Enable inversion IO value of output channel number 0 and make the setting persistent.

```
LucidIoCtrl -dtcp:192.168.177.240:50001 -c0 -soutDiInverted=on -p
```

Read inversion IO value flag of output channel number 0

```
LucidIoCtrl -dtcp:192.168.177.240:50001 -c0 -goutDiInverted
->  outDiInverted=on
```

##### 4.2.2.5.2.3.2 outDiCanCancel

FRAME protocol access to IO configuration parameter *outDiCanCancel*. (→ 4.2.2.3.3.2)

| **Parameter** | *outDiCanCancel* | **Address** | 0x1101 (Bit 1) |
|---|---|---|---|
| **Values** | on/off | **Parameter Type** | 1 Bit |
| **LucidIoCtrl Command Line Tool** | | | |
| **Parameter Name** | *outDiCanCancel* | **Parameter Values** | on/off |
| **Call (Set)** | `LucidIoCtrl -dtcp:[ip:port] -c[Channel] -soutDiCanCancel=[Value] {-p} {--default}` | | |
| **Call (Get)** | `LucidIoCtrl -dtcp:[ip:port] -c[Channel] -goutDiCanCancel` | | |

**Tab. 87 IO Configuration Parameter Bit outDiCanCancel (FRAME Protocol and LucidIoCtrl)**

#### 4.2.2.5.2.3.3 outDiCanRetrigger

FRAME protocol access to IO configuration parameter *outDiCanRetrigger*. (→4.2.2.3.3.3).

| **Parameter** | *outDiCanRetrigger* | **Address** | 0x1101 (Bit 0) |
|---|---|---|---|
| **Values** | on/off | **Parameter Type** | 1 Bit |
| **LucidIoCtrl Command Line Tool** | | | |
| **Parameter Name** | *outDiCanRetrigger* | **Parameter Values** | on/off |
| **Call (Set)** | `LucidIoCtrl –dtcp:[ip:port] –c[Channel] –soutDiCanRetrigger=[Value] {-p} {--default}` | | |
| **Call (Get)** | `LucidIoCtrl –dtcp:[ip:port] –c[Channel] –goutDiCanRetrigger` | | |

**Tab. 88 IO Configuration Parameter Bit outDiCanRetrigger (FRAME Protocol and LucidIoCtrl)**

LucidIoCtrl Command Line Tool Example

Enable output retrigger option of digital output channel number 0 and make the setting persistent.

```
LucidIoCtrl –dtcp:192.168.177.240:50001 –c0 –soutDiCanRetrigger=on –p
```

Read output retrigger option of digital output channel number 0.

```
LucidIoCtrl –dtcp:192.168.177.240:50001 –c0 –goutDiCanRetrigger
->  outDiCanRetrigger=on
```

#### 4.2.2.5.2.4 outDiCycleTime

FRAME protocol access to IO configuration parameter *outDiCycleTime* (→ 4.2.2.3.4)

| **Parameter** | *outDiCycleTime* | **Address** | 0x1110 |
|---|---|---|---|
| | | **Parameter Type** | 4 Bytes unsigned |
| **LucidIoCtrl Command Line Tool** | | | |
| **Parameter Name** | *outDiCycleTime* | **Parameter Values** | Time [µs] |
| **Call (Set)** | `LucidIoCtrl –dtcp:[ip:port] –c[Channel] –soutDiCycleTime=[Time] {-p} {--default}` | | |
| **Call (Get)** | `LucidIoCtrl –dtcp:[ip:port] –c[Channel] –goutDiCycleTime` | | |

**Tab. 89 IO Configuration Parameter outDiCycleTime (FRAME Protocol and LucidIoCtrl)**

LucidIoCtrl Command Line Tool Example

Set $T_{Cycle}$ of digital output channel number 0 to 1.5 s and make the setting persistent.

```
LucidIoCtrl –dtcp:192.168.177.240:50001 –c0 –soutDiCycleTime=1500000 –p
```

Read $T_{Cycle}$ parameter of digital output channel number 0

```
LucidIoCtrl –dtcp:192.168.177.240:50001 –c0 –goutDiCycleTime
->  outDiCycleTime=1500000
```

The current $T_{Cycle}$ parameter is 1.5 seconds

#### 4.2.2.5.2.5 outDiDutyCycle

FRAME protocol access to IO configuration parameter *outDiDutyCycle* (→ 4.2.2.3.5).

| **Parameter** | *outDiDutyCycle* | **Address** | 0x1111 |
|---|---|---|---|
| | | **Parameter Type** | 2 Bytes unsigned |
| **LucidIoCtrl Command Line Tool** | | | |
| **Parameter Name** | *outDiDutyCycle* | **Parameter Values** | Duty Cycle [‰] |
| **Call (Set)** | `LucidIoCtrl -dtcp:[ip:port] -c[Channel] -soutDiDutyCycle=[Value] {-p} {--default}` | | |
| **Call (Get)** | `LucidIoCtrl -dtcp:[ip:port] -c[Channel] -goutDiDutyCylce` | | |

**Tab. 90 IO Configuration Parameter outDiDutyCycle (FRAME Protocol and LucidIoCtrl)**

LucidIoCtrl Command Line Tool Example

Set duty-cycle of digital output channel number 0 to 20% and make the setting persistent.

```
LucidIoCtrl -dtcp:192.168.177.240:50001 -c0 -soutDiDutyCycle=200 -p
```

Read duty-cycle of digital output channel number 0.

```
LucidIoCtrl -dtcp:192.168.177.240:50001 -c0 -goutDiDutyCycle
->  outDiDutyCycle=200
```

#### 4.2.2.5.2.6 outDiOnDelay

FRAME protocol access to IO configuration parameter *outDiOnDelay* (→ 4.2.2.3.6).

| **Parameter** | *outDiOnDelay* | **Address** | 0x1112 |
|---|---|---|---|
| | | **Parameter Type** | 4 Bytes unsigned |
| **LucidIoCtrl Command Line Tool** | | | |
| **Parameter Name** | *outDiOnDelay* | **Parameter Values** | Time [µs] |
| **Call (Set)** | `LucidIoCtrl -dtcp:[ip:port] -c[Channel] -soutDiOnDelay=[Time]{-p} {--default}` | | |
| **Call (Get)** | `LucidIoCtrl -dtcp:[ip:port] -c[Channel] -goutDiOnDelay` | | |

**Tab. 91 IO Configuration Parameter outDiOnDelay (FRAME Protocol and LucidIoCtrl)**

LucidIoCtrl Command Line Tool Example

Set $T_{OnDelay}$ of digital output channel number 0 to 520 ms and make the setting persistent.

```
LucidIoCtrl -dtcp:192.168.177.240:50001 -c0 -soutDiOnDelay=520000 -p
```

Read $T_{OnDelay}$ parameter of digital output channel number 0.

```
LucidIoCtrl -dtcp:192.168.177.240:50001 -c0 -goutDiOnDelay
->  outDiOnDelay=520000
```

#### 4.2.2.5.2.7 outDiOnHold

FRAME protocol access to IO configuration parameter *outDiOnHold (*→ 4.2.2.3.7).

| **Parameter** | *outDiOnHold* | **Address** | 0x1113 |
|---|---|---|---|
| | | **Parameter Type** | 4 Bytes unsigned |
| **LucidIoCtrl Command Line Tool** | | | |
| **Parameter Name** | *outDiOnHold* | **Parameter Values** | Time [µs] |
| **Call (Set)** | `LucidIoCtrl -dtcp:[ip:port] -c[Channel] -soutDiOnHold=[Time] {-p} {--default}` | | |
| **Call (Get)** | `LucidIoCtrl -dtcp:[ip:port] -c[Channel] -goutDiOnHold` | | |

**Tab. 92 IO Configuration Parameter outDiOnHold (FRAME Protocol and LucidIoCtrl)**

LucidIoCtrl Command Line Tool Example

Set $T_{OnHold}$ of digital output channel number 0 to 1200 ms and make the setting persistent.

```
      LucidIoCtrl –dtcp:192.168.177.240:50001 –c0 –soutDiOnHold=1200000 –p
```

Read $T_{OnHold}$ parameter of digital output channel number 0.

```
      LucidIoCtrl –dtcp:192.168.177.240:50001 –c0 –goutDiOnHold
->    outDiOnHold=1200000
```

#### 4.2.2.6 JSON Interface

This section explains how to access DO8 IO channel value and configuration using the JSON interface (→ 3.3.14).

##### 4.2.2.6.1 IO Channel Value

###### 4.2.2.6.1.1 Read IO Channel Value

JSON protocol reads IO channel values by the command IoGet (→ 5.3.2.4.1).

Example:

Read back the IO channel values 0, 1, 8, and15 of a LIOT16-DO8-I-AO8-4MA20 module with 8 digital outputs and 8 analog 4-20mA outputs.

Request:

```
      {"Cmd":"IoGet",
       "SessionId: 123456789,
       "Values":[{"Id":0, "Type":0},
                 {"Id":1, "Type":0},
                 {"Id":8, "Type":35},
                 {"Id":15, Type":35}]}
```

Response:

```
      {"Status":"OK",
       "Values":[{"Id":0, "Type":0, "Value":true},
                 {"Id":1, "Type":0, "Value":true},
                 {"Id":8, Type":35, "Value":15000000}
                 {"Id":15, Type":35, "Value":7500000}]}
```

The response returns for the two digital output channels IO value of "1", 15mA for the analog output channel 8 and 7.5mA for analog output channel 15.

###### 4.2.2.6.1.2 Write IO Channel Value

JSON protocol reads IO channel values by the command IoSet (→ 5.3.2.4.2).

Example:

Set the IO channel values 0, 1, 8, and 15 of a LIOT16-DO8-I-AO8-4MA20 module.
Digital channels 0 and 1 are set to 1, analog output channel 8 is set to 15mA, channel 15 to 7.5mA.

Request:

```
{"Cmd":"IoSet",
 "SessionId: 123456789,
 "Values":[{,"Id":0,"Type":0, "Value":true},
           {,"Id":1,"Type":0, "Value":true},
           {,"Id":8,"Type":35, "Value":15000000},
           {,"Id":15,"Type":35, "Value":7500000}]}
```

Response:

```
{"Status":"OK"}
```

#### 4.2.2.6.2 IO Channel Configuration

JSON protocol accesses the IO channel configuration parameters by commands IoCfgSet (→ 5.3.2.4.3) and IoCfgGet (→ 5.3.2.4.3).

| | |
|---|---|
| "DO":{<br>"Mode": *modeByte*,<br>"Cancel": *cancelFlag*,<br>"Retrig": *retriggerFlag*,<br>"Invert": *invertFlag*,<br>"CycleTime": *cycleTime*,<br>"Duty": *dutyCycle*,<br>"OnTime": *onTime*,<br>"OnDly": *onDelay*} | |

**Tab. 93 Digital Output Channel Configuration JSON Object**

| Name | Type | Description |
|---|---|---|
| Mode | Number | IO Configuration Parameter *outDiMode* (→ 4.2.2.3.2)<br>Parameter value Mode Byte in Tab. 84. |
| Cancel | Boolean | IO Configuration Parameter *outDiCanCancel* (→ 4.2.2.3.3.2) |
| Retrig | Boolean | IO Configuration Parameter *outDiCanRetrigger* (→ 4.2.2.3.3.3) |
| Invert | Boolean | IO Configuration Parameter *outDiInverted* (→ 4.2.2.3.3.1) |
| CycleTime | Number | IO Configuration Parameter *outDiCycleTime* (→ 4.2.2.3.4) |
| Duty | Number | IO Configuration Parameter *outDiDutyCycle* (→ 4.2.2.3.5) |
| OnTime | Number | IO Configuration Parameter *outDiOnHold* (→ 4.2.2.3.7) |
| OnDly | Number | IO Configuration Parameter *outDiOnDelay* (→ 4.2.2.3.6) |

**Tab. 94 Digital Output Channel Configuration JSON Object Values**

##### 4.2.2.6.2.1 Read Configuration

The JSON command IoCfgGet (→ 5.3.2.4.3) reads the IO channel configuration parameters of an IO channel.

Example

The device LIOT-DO8-I-DO8-I has 16 digital output channels.

The configuration of the channel numbers 0 and 11 are read:

Request

```
{"Cmd": "IoCfgGet",
 "SessionId": 123456789,
 "ValueIds":[0,11]}
```

Response

```
{"Status": "OK",
 "Configs":[
  "Id": 0,
  "Type": 1,
  "DO":{
   "Mode": 1,
   "Cancel": false,
   "Retrig": false,
   "Invert": false,
   "CycleTime": 1000000,
   "Duty": 500,
   "OnTime": 1000000,
   "OnDly": 1000000 }},{
  "Id": 11,
  "Type": 1,
  "DO":{
   "Mode": 1,
   "Cancel": false,
   "Retrig": false,
   "Invert": false,
   "CycleTime": 1000000,
   "Duty": 500,
   "OnTime": 1000000,
   "OnDly": 1000000 }}]}
```

The response returns the IO configuration of channel numbers 0 and 11. They operate both in reflect mode with IO configuration parameters *outDiCycleTime* = 1s, *outDiDutyCycle* = 50%, *outDiOnDelay* = 1s and *outDiOnHold* = 1s

#### 4.2.2.6.2.2 Update Configuration

The JSON command IoCfgSet (→ 5.3.2.4.3) writes the IO channel configuration parameters of an IO channel.

If a protocol field is missing, the related IO configuration parameter remains unchained.

Example

The device LIOT-DO8-I-DO8-I has 16 digital output channels.
The configuration of the channel numbers 0 and 11 are updated:

Parameters *outDiCycleTime* is set to 2s, *outDiDutyCycle* to 25%.
All other parameters should remain unchanged.

Request

```
{"Cmd": "IoCfgSet",
 "SessionId": 123456789,
 "Configs":[{
  "Id": 0,
```

```
"Type": 1,
"DO":{
 "CycleTime": 2000000,
 "Duty" 250 }},{
"Id": 11,
"Type": 1,
"DO":{
 "CycleTime": 2000000,
  "Duty" 250 }}]}
```

#### 4.2.2.7 MQTT Client

The digital output channel IO values and configuration are accessible by the MQTT client (→ 3.3.10).

The IO channel values are selected by value identifiers (→ 3.3.10.4.1). They are formatted according to the value format settings (Tab. 95).

| Mode | Value Formats |
|---|---|
| All Modes | Digital On / Off<br>Digital 1 / 0 (Auto Format)<br>Digital true / false |

**Tab. 95 Digital Output Channel Value Formats**

| Mode | IO Config. Param. | Value Identifier |
|---|---|---|
| on-off | outDiCycleTime | Param 0 |
| | outDiDutyCycle | Param 1 |
| duty-cycle | outDiOnDelay | Param 0 |
| | outDiOnHold | Param 1 |

**Tab. 96 Digital Output Channel Configuration Value Identifiers**

(Tab. 96) lists the IO channel configuration parameters, which are accessible by value identifiers.

#### 4.2.2.8 Modbus/TCP Server

The channel IO values and configuration parameters are accessible by Modbus/TCP registers (→ 3.3.11.2).

| Mode | IO Config. Param. | Modbus Register |
|---|---|---|
| on-off | *outDiCycleTime* | Param 0 |
| | *outDiDutyCycle* | Param 1 |
| duty-cycle | *outDiOnDelay* | Param 0 |
| | *outDiOnHold* | Param 1 |

**Tab. 97 Digital Output IO Channel Configuration Modbus/TCP Registers**

Tab. 97 lists the IO configuration parameters, which are accessibly by Modbus/TCP registers.

#### 4.2.2.9 Data Logging

The digital output channel IO values and configuration can be logged (→ 3.4.3).

The IO channel values are selected by value identifiers (→ 3.3.10.4.1). They are formatted according to the value format settings (→ 3.3.10.4.4, Tab. 95).

### 4.2.3 Analog Input (AI8 Function Class)

The AI8 function class acquires the voltage or current values of 8 analog input signals.

| Function Class | Value | Channels |
|---|---|---|
| AI8 | 0x0110 | 8 |

**Tab. 98 Analog Input Function Class**

| Function Class Type | Value | Input Signal Range | |
|---|---|---|---|
| | | $V_{Min}$, $I_{Min}$ | $V_{Max}$, $I_{Max}$ |
| 5 | 0x1000 | 0V | 5V |
| 10 | 0x1001 | 0V | 10V |
| 24 | 0x1005 | 0V | 24V |
| 10S | 0x1011 | -10V | 10V |
| 0M20 | 0x1100 | 0mA | 20mA |

**Tab. 99 Analog Input Function Class Types**

Tab. 98 and Tab. 99 list the AI8 function class and it's supported function class types.

#### 4.2.3.1 IO Signal

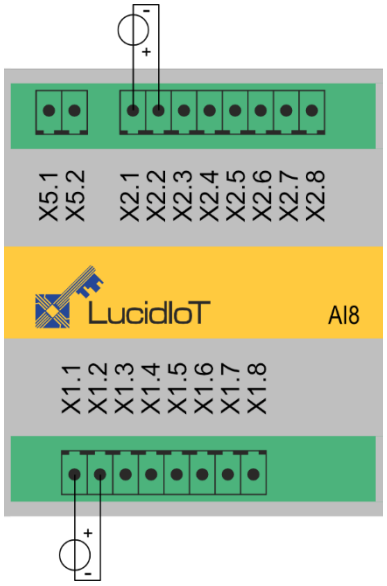



**Fig. 63 AI8 at IO Slot 0**

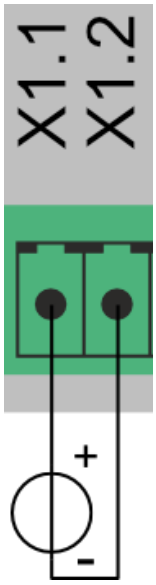



**Fig. 64 AI8 Voltage Signal**

Fig. 63 shows the IO terminals of the AI8 function class installed at IO Slot 0.

Two voltage sources are connected to X1.1 and X1.2 (AI0) and X2.1 and X2.2 (AI4).

Fig. 64 shows the voltage source connected to the IO terminals X1.1 and X1.2 (AI0) in detail.

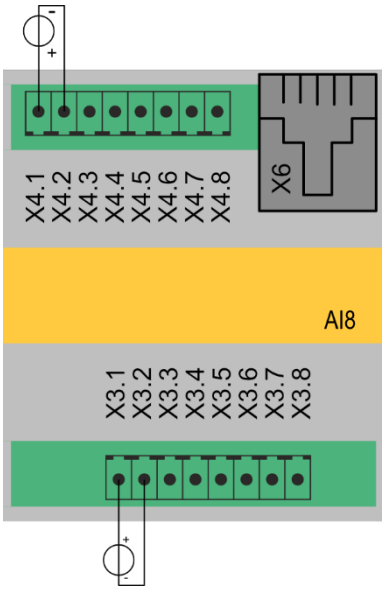



Fig. 65 shows the IO terminals of the AI8 function class installed at IO Slot 1.

Two voltage sources are connected to X3.1 and X3.2 (AI0) and X4.1 and X4.2 (AI4).

**Fig. 65 AI8 at IO Slot 1**

Tab. 100 lists the IO terminals of the AI8 function class.

| IO Terminal | IO Slot | Signal | IO Channel Number |
|---|---|---|---|
| X1.1 | 0 | AI0 + | 0 |
| X1.2 | | AI0 - | |
| X1.3 | | AI1 + | 1 |
| X1.4 | | AI1 - | |
| X1.5 | | AI2 + | 2 |
| X1.6 | | AI2 - | |
| X1.7 | | AI3 + | 3 |
| X1.8 | | AI3 - | |
| X2 | | AI4 ... AI7 | 4 – 7 |
| X3 | 1 | AI0 ... AI3 | 8 – 11 |
| X4 | | AI4 ... AI7 | 12 – 15 |

**Tab. 100 AI8 IO Terminal Connector**

All applied signals must be in the supported range. Under no circumstances, the applied signals must exceed +30 V resp. -30 V.

#### 4.2.3.1.1 Current Inputs

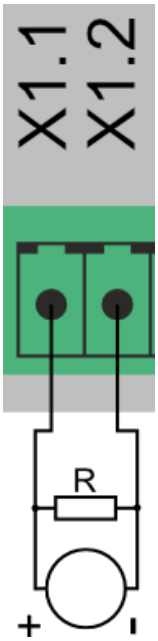



Fig. 66 shows a current source connected to analog input channel 0.

The AI8-0M20 function class is supplied with 8 pcs of 500 Ω precision burden resistors. The resistors are connected to the input terminals in parallel to the input signal.

The AI8-0M20 function class is able to measure both, 0-10 V signals and 0-20 mA signals. The burden resistor is only necessary for 0-20mA current inputs.

The burden resistor transforms the 0-20mA current into a voltage:

$$U_{In} = R \,||R_{In} * I$$

In this formula is I the measured current. R is the 500 Ω burden resistor.

**Fig. 66 AI8 Current Signal**

Most of the current I flows through R but a small part of it flows through the input resistance of the analog input circuit. This causes that the maximum measured voltage is lower than

$$U_{In} = 500\Omega * 20\text{mA} = 10\text{V}$$

The maximum voltage at a current of 20mA is 9,985V caused by the input resistance of 300kΩ.

The current value types (e.g. CUS4) consider the input resistance and return the corrected result.

### 4.2.3.2 Operation

The signals of the active analog input channels are measured subsequently by a 12 Bit ADC.

Analog input channels operating in standard mode are acquired subsequently. The time $T_{AdcScan}$ defines the time within each analog input channel is measured one time.

$T_{AdcScan}$ depends on the number of active input channels and the number of oversampling rounds.

#### 4.2.3.2.1 Operation Modes

##### 4.2.3.2.1.1 Standard

In standard mode, the analog input channels are acquired as explained.

##### 4.2.3.2.1.2 Inactive

In inactive mode, the analog input channel processing is suspended. This improves the processing speed of other active channels.

Reading the IO value returns 0.

#### 4.2.3.2.2 Offset Compensation

The IO configuration parameter *inAnOffset* (→ 4.2.3.3.3) allows compensation of an offset value by adding a value to the measured channel IO value.

#### 4.2.3.2.3 Range Overflow Detection

The analog input channels are able to measure signals, which are a bit outside the specified measurement range. This allows detection of measurement range overflow to the upper and lower signal limits. The margin reserved for this purpose is approx. 1% of the measurement range.

Range overflow detection can be activated for an analog input channel by setting the IO channel configuration flag *inAnOverflow* (→ 4.2.3.3.2.2).

| Value Type | Condition | Returned value |
|---|---|---|
| VOS4 | $V_{In} < V_{Min}$ | 0x80000000 |
| | $V_{In} > V_{Max}$ | 0x7FFFFFFF |
| CUS4 | $I_{In} < I_{Min}$ | 0x80000000 |
| | $I_{In} > I_{Max}$ | 0x7FFFFFFF |

**Tab. 101 Analog Input Channel Overflow Limits**

Tab. 101 lists the limits and the IO values in case of signal exceeding upper and lower measurement limits.

If the flag *inAnOverflow* is disabled, in case of a signal overflow the IO value returns the measured value even if it is outside the expected limits. The developer should consider this value as overflow indication but not rely on its correctness.

Example:
The IO configuration parameter flag *inAnOverflow* is disabled for an AI8-10 input channel.

If a voltage of 11V is applied the analog input channel, it is measured e.g. as 10.1V (reserved margin). This should be considered as a measurement value overflow condition.

If a voltage of -1V is applied to the analog input channel, it is measured e.g. as -0.1V. This should be considered as a measurement value underflow condition.

#### 4.2.3.2.4 Oversampling and Averaging

The 12 Bit ADC accumulates 1024 samples and calculates a precise IO value.

If the flag *inAnAverage* (→ 4.2.3.3.2.1) is set, the 1024 samples are averaged and stored in the IO value.

If the flag *inAnAverage* is not set, the 1024 samples are used in order to calculate an oversampled 16 Bit result.

### 4.2.3.3 Configuration Parameters

#### 4.2.3.3.1 inAnMode

The IO configuration parameter *inAnMode* specifies the operation mode (→ 4.2.3.2.1) of the analog input channel.

| **Parameter** | *inAnMode* | **Access** | Read / Write |
|---|---|---|---|
| **Values** | Input operation mode<br>inactive, standard | | |
| **Default Value** | standard | **Parameter Type** | - |

**Tab. 102 IO Channel Configuration Parameter inAnMode**

#### 4.2.3.3.2 inAnFlags

The IO configuration parameter *inAnFlags* specifies the operation flags of the analog input channel.

##### 4.2.3.3.2.1 inAnAverage

The IO configuration parameter *inAnAverage* specifies if the measured IO channel value is averaged of oversampled. (→ 4.2.3.2.4)

| **Parameter** | *inAnAverage* | **Access** | Read / Write |
|---|---|---|---|
| **Values** | on/off | | |
| **Default Value** | off | **Parameter Type** | 1 Bit |

**Tab. 103 IO Channel Configuration Parameter Bit inAnAverage**

##### 4.2.3.3.2.2 inAnOverflow

The IO configuration parameter *inAnOverflow* specifies function of value overflow detection. (→ 4.2.3.2.3)

| **Parameter** | *inAnOverflow* | **Access** | Read / Write |
|---|---|---|---|
| **Values** | on/off | | |
| **Default Value** | off | **Parameter Type** | 1 Bit |

**Tab. 104 IO Channel Configuration Parameter Bit inAnOverflow**

#### 4.2.3.3.3 inAnOffset

The IO configuration parameter *inAnOffset* specifies an offset compensation value in mV / mA. The offset is added to the IO value.

| **Parameter** | *inAnOffset* | **Access** | Read / Write |
|---|---|---|---|
| **Values** | Offset in mV / µA<br>-3V ~ 3V / -3mA ~ 3mA | | |
| **Default Value** | 0 | **Parameter Type** | |

**Tab. 105 IO Channel Configuration Parameter inAnOffset**

### 4.2.3.4 Web Interface

This section explains how the AI8 IO channel value and configuration are accessed using the web interface.

#### 4.2.3.4.1 IO Channel Value

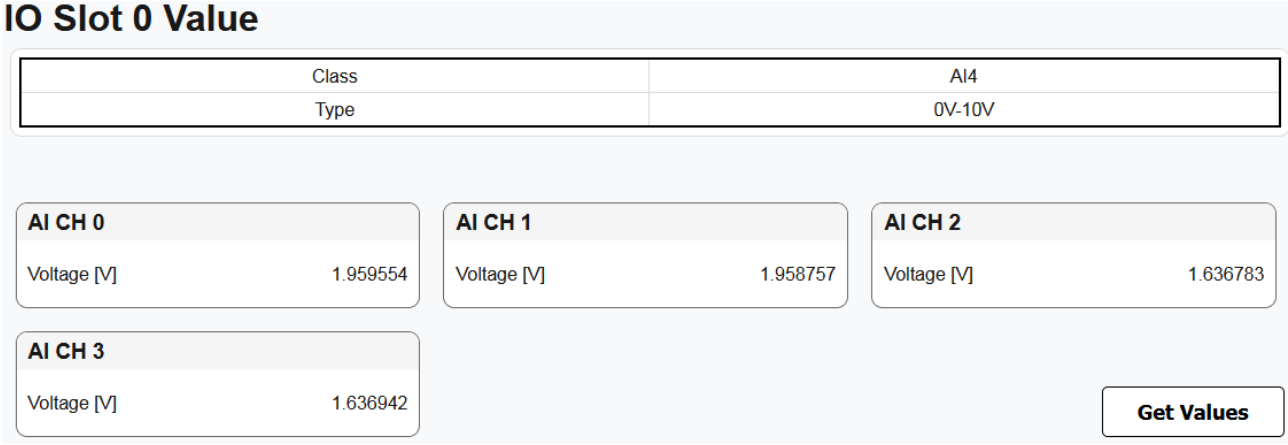



**Fig. 67 Analog Input Channel Value Access**

Fig. 67 shows the IO Slot 0 value page of the web interface for the analog input channels.

Only users with IO Value access rights are allowed to access this page. It is not visible for other users.

The IO values of all channels are read by clicking the *Get Values* button.

#### 4.2.3.4.2 IO Channel Configuration

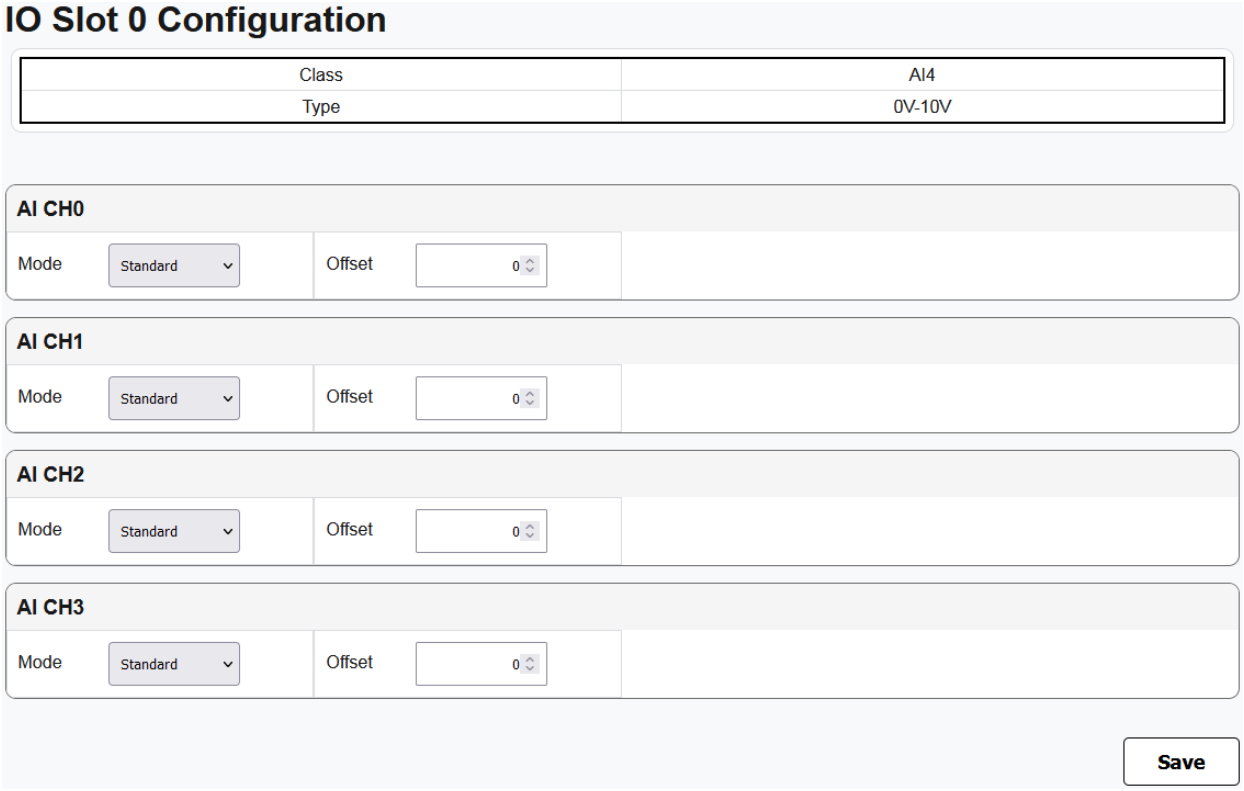



**Fig. 68 Analog Input Channel Configuration**

Fig. 68 shows the IO Slot 0 configuration page of the analog input channels.

Only users with IO Config access rights are allowed to access this page. It is not visible for other users.

The IO Configuration parameters (→ 4.2.3.3) are updated and permanently saved by clicking the *Save* button.

### 4.2.3.5 FRAME Interface

This section explains how the AI8 function class values and configuration parameters are accessed by using the FRAME interface (→ 5.4) and the LucidIoCtrl command line tool.

#### 4.2.3.5.1 IO Channel Value

The supported IO value type (→ 5.2) depends on the type of the analog input function.

| Function Class | Supported FRAME Value Type |
|---|---|
| Voltage Input | VOS4 |
| Current Input | CUS4 |
| | VOS4 |

**Tab. 106 Analog Input IO Value Types**

#### 4.2.3.5.1.1 FRAME GetIo Command

The FRAME protocol command GetIo (→ 5.4.4.1) reads and returns the value of an IO channel number.

| **Command** | GetIo | **Access** | Read |
|---|---|---|---|
| **Opcode** | 0x46 | | |
| **LucidIoCtrl Arguments** | | | |
| **Call (-tV)** | `LucidIoCtrl -dtcp:[ip:port] -c[Channel] -tV -r` | | |
| **Return** | `CHn:v`<br>n: IO Channel Number<br>v: Analog Input Voltage [V] | | |
| **Call (-tC)** | `LucidIoCtrl -dtcp:[ip:port] -c[Channel] -tC -r` | | |
| **Return** | `CHn:v`<br>n: IO Channel Number<br>v: Analog Input Current [mA] | | |

**Tab. 107 Analog Input FRAME GetIo Command and LucidIoCtrl Arguments**

LucidIoCtrl Command Line Tool Example

Read voltage from input channel number 0:

```
      LucidIoCtrl -dtcp:192.168.177.240:50001 -c0 -tV -r
->    CH0:5.000
```

Returns voltage of 5V.

Read current from input channel 0:

```
      LucidIoCtrl -dtcp:192.168.177.240:50001 -c0 -tC -r
->    CH0:15.000
```

Returns current of 15mA.

#### 4.2.3.5.1.2 FRAME GetIoGroup Command

The FRAME protocol command GetIoGroup (→ 5.4.4.2) reads and returns the values of a group of IO channel numbers.

<table>
<tr><td>Command</td><td>GetIoGroup</td><td>Access</td><td>Read</td></tr>
<tr><td>Opcode</td><td>0x48</td><td></td><td></td></tr>
<tr><td colspan="4">LucidIoCtrl Command Line Tool</td></tr>
<tr><td>Call (-tV)</td><td colspan="3"><code>LucidIoCtrl –dtcp:[ip:port]–c[Channels] –tV –r</code><br><br><u>Channels:</u><br>Comma separated list of IO channel numbers e.g. -c0,1,3</td></tr>
<tr><td>Return</td><td colspan="3">List of values sorted from lower to higher IO channel numbers<br><code>CHn:v</code><table><tr><td>n</td><td>Input Channel Number</td></tr><tr><td>v</td><td>Analog Input Voltage [V]</td></tr></table></td></tr>
<tr><td>Call (-tC)</td><td colspan="3"><code>LucidIoCtrl –dtcp:[ip:port] –c[Channels] –tC –r</code><br><br><u>Channels:</u><br>Comma separated list of IO channel numbers e.g. -c0,1,3</td></tr>
<tr><td>Return</td><td colspan="3">List of values sorted from lower to higher channels<br><code>CHn:v</code><table><tr><td>n</td><td>IO Channel Number</td></tr><tr><td>v</td><td>Analog Input Current [mA]</td></tr></table></td></tr>
</table>

**Tab. 108 Analog Input FRAME GetIoGroup command and LucidIoCtrl Arguments**

<u>LucidIoCtrl Command Line Tool Example</u>

Read voltages of analog input (and output) channel numbers 0 to 3.

```
      LucidIoCtrl –dtcp:192.168.177.240:50001 –c0,1,2,3 –tV –r
->    CH0:6.000  CH1:2.500  CH2:0.000  CH3:-2.500
```

### 4.2.3.5.2 IO Channel Configuration

IO channel configuration parameters are set by the FRAME protocol command SetParam (→ 5.4.4.5).

IO channel configuration parameters are read by the FRAME protocol command GetParam (→ 5.4.4.6).

#### 4.2.3.5.2.1 inAnMode

FRAME protocol access to *inAnMode* IO configuration parameter (→ 4.2.3.3.1).

<table>
<tr><td>Parameter</td><td>inAnMode</td><td>Address</td><td>0x1100</td></tr>
<tr><td>Values</td><td colspan="3">Input Mode
<table>
<tr><th>Mode</th><th>Byte</th><th>Mode Value</th></tr>
<tr><td>Inactive</td><td>0x00</td><td>inactive</td></tr>
<tr><td>Standard</td><td>0x01</td><td>standard</td></tr>
</table></td></tr>
<tr><td></td><td></td><td>Parameter Type</td><td>1 Byte unsigned</td></tr>
<tr><td colspan="4">LucidIoCtrl Command Line Tool</td></tr>
<tr><td>Parameter Name</td><td>inAnMode</td><td>Parameter Values</td><td>Mode Value</td></tr>
<tr><td>Call (Set)</td><td colspan="3"><code>LucidIoCtrl –dtcp:[ip:port] –c[Channel]<br>–sinAnMode=[Mode Value] {-p} {--default}</code></td></tr>
<tr><td>Call (Get)</td><td colspan="3"><code>LucidIoCtrl –dtcp:[ip:port]–c[Channel] –ginAnMode</code></td></tr>
</table>

**Tab. 109 IO Configuration Parameter inAnMode (FRAME Protocol and LucidIoCtrl)**

LucidIoCtrl Command Line Tool Example

Set operation mode of input channel number 0 to standard dode and make the setting persistent.

```
LucidIoCtrl –dtcp:192.168.177.240:50001 –c0 –sinAnMode=standard –p
```

Read the operation mode of input channel number 0

```
LucidIoCtrl –dtcp:192.168.177.240:50001 –c0 –ginAnMode
->  inAnMode=standard
```

#### 4.2.3.5.2.2 inAnFlags

FRAME protocol access to IO configuration parameter *inAnFlags* (→ 4.2.3.3.2).

The IO configuration parameter *inAnFlags* cannot be accessed by LucidIoCtrl directly. Use the bit parameters instead.

<table>
<tr><td>Parameter</td><td>inAnFlags</td><td>Address</td><td>0x1101</td></tr>
<tr><td>Values</td><td colspan="3">Consists of the following Bit Parameters
<table>
<tr><th>Bit Parameter</th><th>Bit Postion</th></tr>
<tr><td>inAnAverage</td><td>Bit 0</td></tr>
<tr><td>inAnOverflow</td><td>Bit 1</td></tr>
</table></td></tr>
<tr><td>Default Value</td><td>0x00</td><td>Parameter Type</td><td>1 Byte unsigned</td></tr>
</table>

**Tab. 110 IO Configuration Parameter inAnFlags (FRAME Protocol)**

##### 4.2.3.5.2.2.1 inAnAverage

FRAME protocol access to IO configuration parameter *inAnAverage* (→4.2.3.3.2.1).

| **Parameter** | *inAnAverage* | **Address** | 0x1101 (Bit 0) |
|---|---|---|---|
| **Values** | on/off | **Parameter Type** | 1 Bit |
| **LucidIoCtrl Command Line Tool** | | | |
| **Parameter Name** | *inAnAverage* | **Parameter Values** | on/off |
| **Call (Set)** | `LucidIoCtrl -dtcp:[ip:port]-c[Channel] -sinAnAverage=[Value] {-p} {--default}` | | |
| **Call (Get)** | `LucidIoCtrl -d[COMx] -c[Channel] -ginAnAverage` | | |

**Tab. 111 IO Configuration Parameter Bit inAnAverage (FRAME Protocol and LucidIoCtrl)**

#### 4.2.3.5.2.2.2 inAnOverflow

FRAME protocol access to IO configuration parameter *inAnOverflow* (→ 4.2.3.3.2.2).

| **Parameter** | *inAnOverflow* | **Address** | 0x1101 (Bit 1) |
|---|---|---|---|
| **Values** | on/off | **Parameter Type** | 1 Bit |
| **LucidIoCtrl Command Line Tool** | | | |
| **Parameter Name** | *inAnOverflow* | **Parameter Values** | on/off |
| **Call (Set)** | `LucidIoCtrl -dtcp:[ip:port] -c[Channel] -sinAnOverflow=[Value] {-p} {--default}` | | |
| **Call (Get)** | `LucidIoCtrl -dtcp:[ip:port] -c[Channel] -ginAnOverflow` | | |

**Tab. 112 IO Configuration Parameter Bit inAnOverflow (FRAME Protocol and LucidIoCtrl)**

### 4.2.3.5.3 inAnOffset

FRAME protocol access to IO configuration parameter *inAnOffset* (→ 4.2.3.3.3).

| **Parameter** | *inAnOffset* | **Address** | 0x1120 |
|---|---|---|---|
| | | **Parameter Type** | 2 Bytes signed |
| **LucidIoCtrl Command Line Tool** | | | |
| **Parameter Name** | *inAnOffset* | **Parameter Values** | Offset [mV / µA] |
| **Call (Set)** | `LucidIoCtrl -dtcp:[ip:port] -c[Channel] -sinAnOffset=[Value] {-p} {--default}` | | |
| **Call (Get)** | `LucidIoCtrl -dtcp:[ip:port] -c[Channel] -ginAnOffset` | | |

**Tab. 113 IO Configuration Parameter inAnOffset (FRAME Protocol and LucidIoCtrl)**

LucidIoCtrl Command Line Tool Example

Set offset compensation value of input channel number 0 to -5mV and make the setting persistent.

```
      LucidIoCtrl -dtcp:192.168.177.240:50001 -c0 -sinAnOffset=-5 -p
```

Read offset compensation value

```
      LucidIoCtrl -dtcp:192.168.177.240:50001 -c0 -ginAnOffset
->    inAnOffset=-5
```

### 4.2.3.6 JSON Interface

This section explains how to access IO channel value and configuration using the JSON interface (→ 3.3.14).

#### 4.2.3.6.1 IO Channel Value

##### 4.2.3.6.1.1 Read IO Channel Value

JSON protocol reads IO channel values by the command IoGet (→ 5.3.2.4.1).

Example:
Read IO channel values 0, 1, 8, and15 of a LIOT16-DO8-I-AI8-20MA module with 8 digital outputs and 8 analog 0-20mA (0-10V) inputs.

Request:
```
{"Cmd":"IoGet",
 "SessionId: 123456789,
 "Values":[{"Id":0, "Type":0},
           {"Id":1, "Type":0},
           {"Id":8, "Type":35},
           {"Id":15, Type":35}]}
```

Response:
```
{"Status":"OK",
 "Values":[{"Id":0, "Type":0, "Value":true},
           {"Id":1, "Type":0, "Value":true},
           {"Id":8, Type":29, "Value":5000000}
           {"Id":15, Type":35, "Value":7500000}]}
```
The response returns for the two digital output channels IO value of "1", 5V for the analog input channel 8 and 7.5mA for analog input channel 15.

#### 4.2.3.6.2 Channel Configuration

JSON protocol accesses the IO channel configuration parameters by commands IoCfgSet (→ 5.3.2.4.3) and IoCfgGet (→ 5.3.2.4.3).

| | |
|---|---|
| "AI":{<br>"Mode": *mode*,<br>"Average": *average*,<br>"Overflow": *overflow*,<br>"Offset": *offset*} | |

**Tab. 114 Analog Input Channel Configuration JSON Object**

| Name | Type | Description |
|---|---|---|
| Mode | Number | IO Configuration Parameter *inAnMode* (→ 4.2.3.3.1)<br>Parameter value Mode Byte in Tab. 109. |
| Average | Boolean | IO Configuration Parameter *inAnAverage* (→ 4.2.3.3.2.1) |
| Overflow | Boolean | IO Configuration Parameter *inAnOverflow* (→ 4.2.3.3.2.2) |
| Offset | Number | IO Configuration Parameter *inAnOffset* (→ 4.2.3.3.3) |

**Tab. 115 Analog Input Channel Configuration JSON Object Values**

Fields of the analog input channel configuration JSON object are optional.

#### 4.2.3.6.2.1 Read Configuration

The JSON command IoCfgGet (→ 5.3.2.4.3) reads the IO channel configuration parameters of an IO channel.

Example
The device LIOT-AI8-10-AO8-10 has 8 analog 10V input channels (Channel numbers 0 to 7) and 8 analog 10V output channels (Channel numbers 8 to 15f).

The configuration of the channel numbers 0 and 3 are read:

Request

```
{"Cmd": "IoCfgGet",
 "SessionId": 123456789,
 "ValueIds":[0,3]}
```

Response

```
{"Status": "OK",
 "Configs":[
  "Id": 0,
  "Type": 2,
  "AI":{
   "Mode": 1,
   "Average": false,
   "Overflow": false,
   "Offset": 0 }},{
  "Id": 3,
  "Type": 2,
  "AI":{
   "Mode": 1,
   "Average": true,
   "Overflow": true,
   "Offset": -1000 }}]}
```

The response returns the IO configuration parameters of channel numbers 0 and 3. Both analog input channels are active. For channel number 0 averaging of the result and overflow detection are disabled, for channel number 3 both options are enabled. Channel number 3 subtracts an offset of 1V (or 1mA) from the acquired result.

#### 4.2.3.6.2.2 Update Configuration

The JSON command IoCfgSet (→ 5.3.2.4.3) writes the IO channel configuration parameters of an IO channel.

If an IO configuration parameter is not included in the object, it remains unchanged.

Example
The device LIOT-AI8-10-AO8-10 has 8 analog 10V input channels (Channel numbers 0 to 7) and 8 analog 10V output channels (Channel numbers 8 to 15).

Overflow detection and averaging should be disabled for channel 0, an offset of -1V should be set for channel 3. All other parameters should remain unchanged.

Request

```
{"Cmd": "IoCfgSet",
 "SessionId": 123456789,
 "Configs":[{
  "Id": 0,
  "Type": 2,
  "AI":{
   "Average": true,
   "Overflow" false }},{
  "Id": 3,
  "Type": 2,
  "AI":{
   "Offset": -1000 }}]}
```

#### 4.2.3.7 MQTT Client

The IO channel values are accessible by the MQTT client (→ 3.3.10).

The IO channel values are selected by value identifiers (→ 3.3.10.4.1). They are formatted according to the value format settings (Tab. 116).

| Function Class Type | Value Formats |
|---|---|
| 5, 10, 24, 10S | Voltage (µV) Integer (Auto)<br>Voltage (V) Decimal<br>Voltage (V) Decimal Prec 0<br>Voltage (V) Decimal Prec 1<br>Voltage (V) Decimal Prec 2<br>Voltage (V) Decimal Prec 3 |
| 0M20 | Voltage (µV) Integer (Auto)<br>Voltage (V) Decimal<br>Voltage (V) Decimal Prec 0<br>Voltage (V) Decimal Prec 1<br>Voltage (V) Decimal Prec 2<br>Voltage (V) Decimal Prec 3<br>Current (nA) Integer<br>Current (mA) Decimal<br>Current (mA) Decimal Prec 0<br>Current (mA) Decimal Prec 1<br>Current (mA) Decimal Prec 2<br>Current (mA) Decimal Prec 3 |

**Tab. 116 Analog Input Channel Value Formats**

#### 4.2.3.8 Modbus/TCP Server

The IO channel values are accessible by Modbus/TCP registers (→ 3.3.11.2).

32 bit IO value registers contain the voltage in µV, 16 bit IO value registers in mV.

For 0M20 (current) function class, currents are accessible by alternative IO value registers. 32 bit alternative IO value registers contain the current in nA, 16 bit registers in µA.

IO value registers are in signed format.

#### 4.2.3.9 Data Logging

The IO channel values can be logged (→ 3.4.3).

The IO channel values are selected by value identifiers (→ 3.3.10.4.1). They are formatted according to the value format settings (→ 3.3.10.4.4, Tab. 116).

## 4.2.4 Analog Output (AO8 Function Class)

The AO8 function class controls 8 analog voltage or current signals.

| Function Class | Value | Channels |
|---|---|---|
| AO8 | 0x1110 | 8 |

**Tab. 117 Analog Output Function Class**

| Function Class Type | Value | Output Signal Range | |
|---|---|---|---|
| | | $V_{Min}$, $I_{Min}$ | $V_{Max}$, $I_{Max}$ |
| 5 | 0x1000 | 0V | 5V |
| 10 | 0x1001 | 0V | 10V |
| 24 | 0x1005 | 0V | 24V |
| 0MA20 | 0x1100 | 0mA | 20mA |
| 4MA20 | 0x1101 | 4mA | 20mA |

**Tab. 118 Analog Output Function Class Types**

Tab. 117 and Tab. 118 list the AO8 function class and it's supported function class types.

### 4.2.4.1 IO Signal

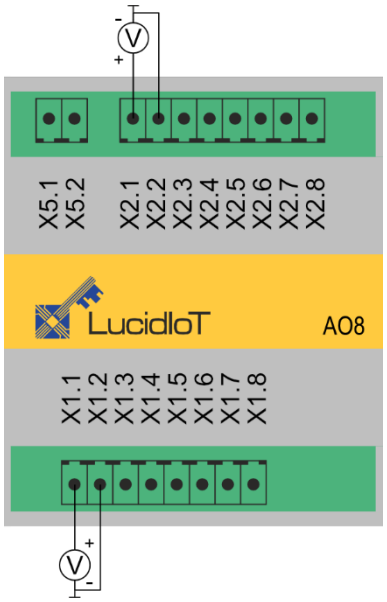



**Fig. 69 AO8 at IO Slot 0**

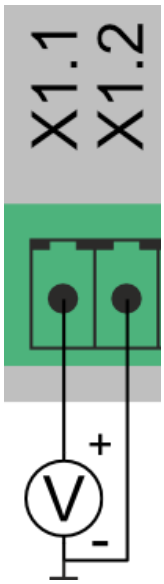



**Fig. 70 AO8 Voltage Signal**

Fig. 69 shows the IO terminals of the AO8 function class installed at IO Slot 0.

Two voltmeters are connected to X1.1 and X1.2 (AO0) and X2.1 and X2.2 (AO4).

Fig. 70 shows the voltmeter connected to the IO terminals X1.1 and X1.2 (AO0) in detail.

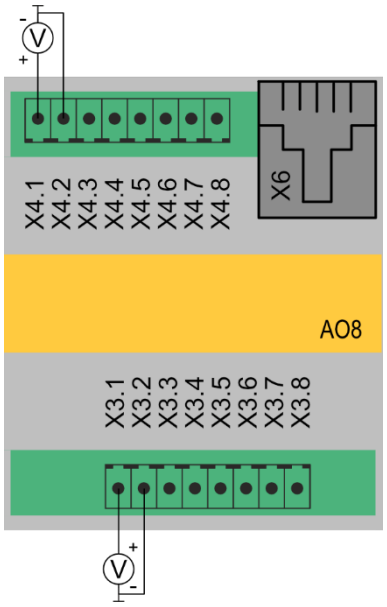



Fig. 71 shows the IO terminals of the AO8 function class installed at IO Slot 1.

Two voltmeters are connected to X3.1 and X3.2 (AO0) and X4.1 and X4.2 (AO4).

**Fig. 71 AO8 at IO Slot 1**

Tab. 120 lists the IO terminals of the AO8 function class.

| IO Terminal | IO Slot | Signal | IO Channel Number |
|---|---|---|---|
| X1.1 | 0 | AO0 | 0 |
| X1.2 | | GND | |
| X1.3 | | AO1 | 1 |
| X1.4 | | GND | |
| X1.5 | | AO2 | 2 |
| X1.6 | | GND | |
| X1.7 | | AO3 | 3 |
| X1.8 | | GND | |
| X2 | | AO4 ... AO7 | 4 – 7 |
| X3 | 1 | AO0 ... AO3 | 8 – 11 |
| X4 | | AO4 ... AO7 | 12 – 15 |

**Tab. 119 AO8 IO Terminal Connector**

#### 4.2.4.1.1 Current Outputs

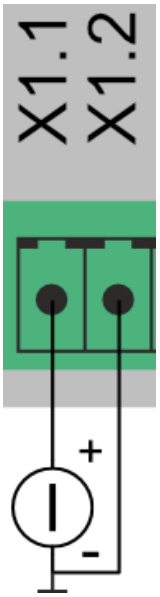



Fig. 72 shows a current meter connected to analog input channel 0 of a 0MA20 or 4MA20 function class.

**Fig. 72 AO8**
**Current Signal**

No external voltage or current must be applied to the terminals.
The load current per channel must not exceed $I_{ChMax}$. The sum of all 8 output channels must not exceed $I_{TotalMax}$.

### 4.2.4.2 Operation

The AO8 function class generates 8 independent output voltages or currents in the range of $V_{Min}$ / $I_{Min}$ and $V_{Max}$ / $I_{Max}$.

#### 4.2.4.2.1 Operation Modes

##### 4.2.4.2.1.1 Standard

In standard mode, the analog output channels are processed as explained.

##### 4.2.4.2.1.2 Inactive

In inactive mode, the analog output channels are constant at $V_{Min}$ / $I_{Min}$ value.

#### 4.2.4.2.2 Offset Compensation

The IO configuration parameter *outAnOffset* (→ 4.2.4.3.2) allows compensation of an offset value by adding a value to the output channel IO value.

### 4.2.4.3 Configuration Parameters

#### 4.2.4.3.1 outAnMode

The IO configuration parameter *outAnMode* specifies the operation mode (→ 4.2.4.2.1) of the analog output channel.

| **Parameter** | *outAnMode* | **Access** | Read / Write |
|---|---|---|---|
| **Values** | Operation mode<br>inactive, standard | | |
| **Default Value** | standard | **Parameter Type** | - |

**Tab. 120 IO Channel Configuration Parameter outAnMode**

#### 4.2.4.3.2 outAnOffset

The IO configuration parameter *outAnOffset* specifies an offset compensation value in mV / mA. The offset is added to the IO value.

| **Parameter** | *outAnOffset* | **Access** | Read / Write |
|---|---|---|---|
| **Values** | Offset in mV / µA<br>-3V ~ 3V / -3mA ~ 3mA | | |
| **Default Value** | 0 | **Parameter Type** | |

**Tab. 121 IO Channel Configuration Parameter outAnOffset**

### 4.2.4.4 Web Interface

#### 4.2.4.4.1 IO Channel Value

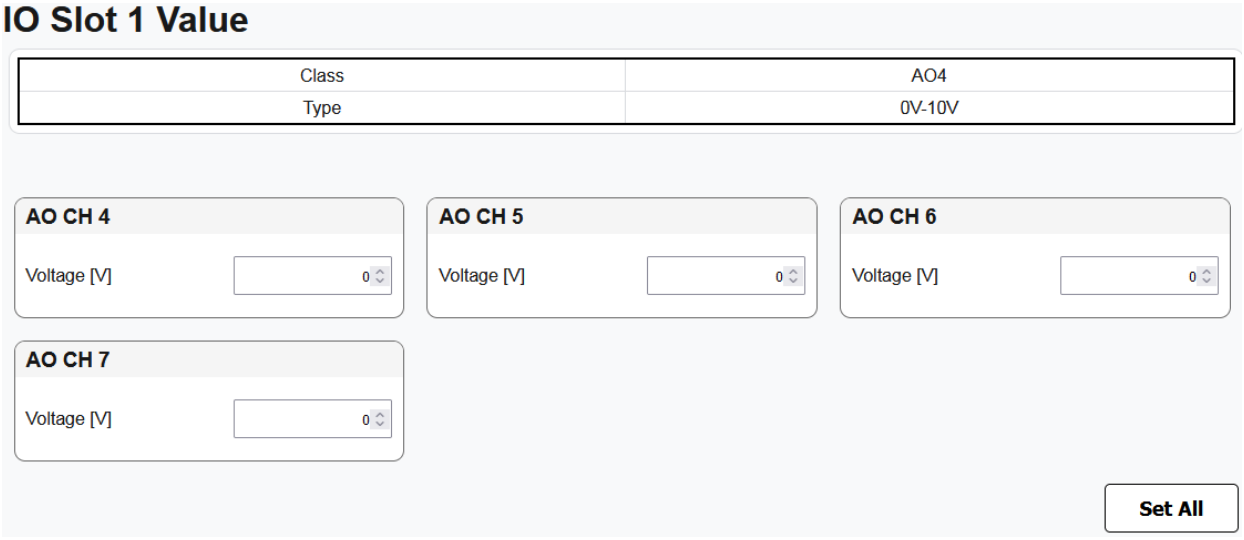



**Fig. 73 Analog Output Channel Value Access**

Fig. 73 shows the IO slot value page for the analog output channel.

Only users with IO Value access rights are allowed to access this page. It is not visible for other users.

Depending the function type values are entered in µV or nA without decimals. All output channel values are set by clicking *Set All* button.

#### 4.2.4.4.2 IO Channel Configuration

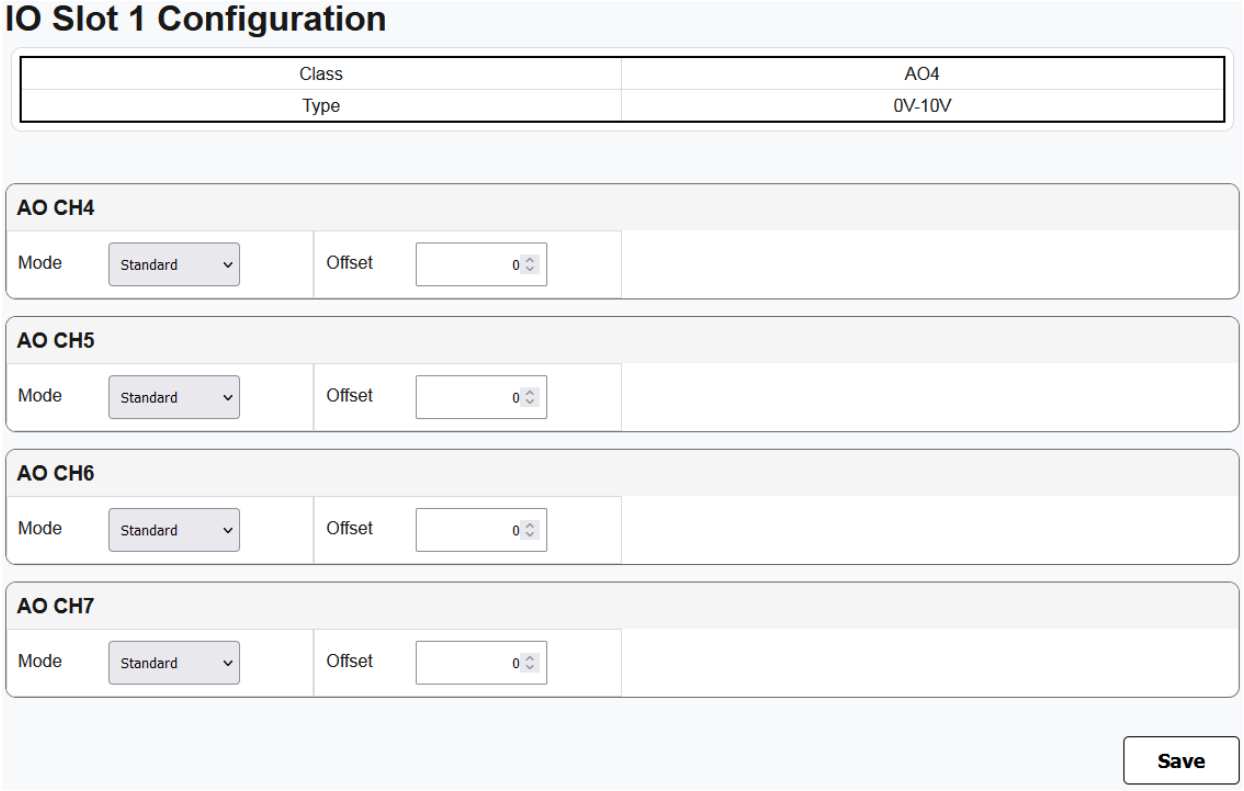



**Fig. 74 Analog Output Channel Configuration**

Fig. 74 shows the IO slot configuration page of analog output channels.

Only users with IO Config access rights are allowed to access this page. It is not visible for other users.

The IO Configuration parameters (→ 4.2.4.3) are updated and permanently saved by clicking the *Save* button.

### 4.2.4.5 FRAME Interface

This section explains how the AO8 function class values and configuration parameters are accessed by using the FRAME interface (→ 5.4) and the LucidIoCtrl command line tool.

#### 4.2.4.5.1 IO Channel Value

The available IO value type (→ 5.2) depends on the type of the analog output function.

| Function Class | Supported FRAME Value Type |
|---|---|
| Voltage Output | VOS4 |
| Current Output | CUS4 |

**Tab. 122 Analog Output IO Value Types**

#### 4.2.4.5.1.1 FRAME GetIo Command

→ 4.2.3.5.1.1

#### 4.2.4.5.1.2 FRAME GetIoGroup Command

→ 4.2.3.5.1.2

#### 4.2.4.5.1.3 FRAME SetIo Command

The FRAME protocol command SetIo (→ 5.4.4.3) writes the value of an IO channel number.

| **Command** | SetIo | **Access** | Write |
|---|---|---|---|
| **Opcode** | 0x40 | | |
| **LucidIoControl Command Line Tool** | | | |
| **Call (-tV)** | `LucidIoCtrl -dtcp:[ip:port] -c[Channel] -tV -w[Voltage]` | | |
| **Call (-tC)** | `LucidIoCtrl -dtcp:[ip:port] -c[Channel] -tC -w[Current]` | | |

**Tab. 123 Analog Output FRAME SetIo Command and LucidIoCtrl Arguments**

LucidIoCtrl Command Line Tool Example

Set output channel number 0 to 2.540 V:

```
LucidIoCtrl -dtcp:192.168.177.240:50001 -c0 -tV -w2.540
```

Set output channel number 0 to 10 mA:

```
LucidIoCtrl -dtcp:192.168.177.240:50001 -c0 -tC -w10
```

#### 4.2.4.5.1.4 FRAME SetIoGroup Command

The FRAME protocol command SetIoGroup (→ 5.4.4.4) writes the values of a group of IO channel numbers.

| **Command** | SetIoGroup | **Access** | Write |
|---|---|---|---|
| **Opcode** | 0x42 | | |
| **LucidIoCtrl Command Line Tool** | | | |
| **Call (-tV)** | `LucidIoCtrl –dtcp:[ip:port] –c[Channels] –tV –w[Voltages]`<br><br>Channels:<br>Comma separated list of channels e.g. –c0,1,3<br><br>Values:<br>Comma separated list of values to set e.g. -w2.5,5,7.5 | | |
| **Call (-tC)** | `LucidIoCtrl –dtcp:[ip:port] –c[Channels] –tC –w[Currents]` | | |

**Tab. 124 Analog Output FRAME SetIoGroup Command and LucidIoCtrl Arguments**

LucidIoCtrl Command Line Tool Example

Set output channel 0 to 1.25 V, output channel 2 to 2.50 V and output channel 3 to 7.50:

```
LucidIoCtrl –dtcp:192.168.177.240:50001 –c0,2,3 –tV –w1.25,2.5,7.5
```

Set output channel 0 to 5 mA, output channel 2 to 15.5 mA and output channel 3 to 20:

```
LucidIoCtrl –dtcp:192.168.177.240:50001 –c0,2,3 –tC –w5,15.5,20
```

#### 4.2.4.5.2 Channel Configuration

IO channel configuration parameters are set by the FRAME protocol command SetParam (→ 5.4.4.5).

IO channel configuration parameters are read by the FRAME protocol command GetParam (→ 5.4.4.6).

##### 4.2.4.5.2.1 outAnMode

FRAME protocol access to *outAnMode* IO configuration parameter (→ 4.2.4.3.1).

| **Parameter** | *outAnMode* | **Address** | 0x1100 |
|---|---|---|---|
| **Values** | Output Mode<br>**Mode** / **Byte** / **Mode Value**<br>Inactive / 0x00 / inactive<br>Standard / 0x01 / standard | | |
| | | **Parameter Type** | 1 Byte unsigned |
| **LucidIoCtrl Command Line Tool** | | | |
| **Parameter Name** | *outAnMode* | **Parameter Values** | Mode Value |
| **Call (Set)** | `LucidIoCtrl –dtcp:[ip:port] –c[Channel] –soutAnMode=[Mode Value] {-p} {--default}` | | |
| **Call (Get)** | `LucidIoCtrl –dtcp:[ip:port] –c[Channel] –goutAnMode` | | |

**Tab. 125 IO Configuration Parameter outAnMode (FRAME Protocol and LucidIoCtrl)**

LucidIoCtrl Command Line Tool Example

Set operation mode of output channel number 0 to standard mode and make the setting persistent.

```
      LucidIoCtrl –dtcp:192.168.177.240:50001 –c0 –soutAnMode=standard –p
```

Read the operation mode of input channel number 0.

```
      LucidIoCtrl –dtcp:192.168.177.240:50001 –c0 –goutAnMode
->    outAnMode=standard
```

#### 4.2.4.5.2.2 outAnOffset

FRAME protocol access to IO configuration parameter *outAnOffset* (→ 4.2.4.3.2).

| **Parameter** | *outAnOffset* | **Address** | 0x1120 |
|---|---|---|---|
| | | **Parameter Type** | 2 Bytes signed |
| **LucidIoCtrl Command Line Tool** | | | |
| **Parameter Name** | *outAnOffset* | **Parameter Values** | Offset [mV / µA] |
| **Call (Set)** | `LucidIoCtrl –dtcp:[ip:port] –c[Channel] –sinAnOffset=[Value] {-p} {--default}` | | |
| **Call (Get)** | `LucidIoCtrl –dtcp:[ip:port] –c[Channel] –ginAnOffset` | | |

**Tab. 126 IO Configuration Parameter outAnOffset (FRAME Protocol and LucidIoCtrl)**

LucidIoCtrl Command Line Tool Example

Set offset compensation value of output channel number 0 to -5mV and make the setting persistent.

```
      LucidIoCtrl –dtcp:192.168.177.240:50001 –c0 –soutAnOffset=-5 –p
```

Read offset compensation value

```
      LucidIoCtrl –dtcp:192.168.177.240:50001 –c0 –goutAnOffset
->    inAnOffset=-5
```

### 4.2.4.6 JSON Interface

This section explains how to access the IO channel value and configuration using the JSON interface (→ 3.3.14).

#### 4.2.4.6.1 IO Channel Value

##### 4.2.4.6.1.1 Read IO Channel Value

JSON protocol reads IO channel values by the command IoGet (→ 5.3.2.4.1).

Example:

Read IO channel values 0, 1, 8, and15 of a LIOT16-DI8-24V-AO8-20MA module with 8 digital 24 V inputs channels and 8 analog 0-20mA output channels.

Request:

```
      {"Cmd":"IoGet",
       "SessionId: 123456789,
       "Values":[{"Id":0, "Type":0},
                 {"Id":1, "Type":0},
                 {"Id":8, "Type":35},
                 {"Id":15, Type":35}]}
```

Response:

```
{"Status":"OK",
 "Values":[{"Id":0, "Type":0, "Value":true},
           {"Id":1, "Type":0, "Value":true},
           {"Id":8, Type":35, "Value":15000000}
           {"Id":15, Type":35, "Value":7500000}]}
```

The response returns for the two digital input channels IO value of "1", 15mA for the analog output channel 8 and 7.5mA for analog output channel 15.

#### 4.2.4.6.1.2 Set IO Channel Value

JSON protocol sets IO channel values by the command IoSet (5.3.2.4.2).

Example:

Set the IO channel values 0, 1, 8, and 15 of a LIOT16-DO8-I-AO8-10V module.
Digital output channels 0 and 1 are set to 1, analog output channel 8 is set to 2.5V, channel 15 to 10V.

Request:

```
{"Cmd":"IoSet",
 "SessionId: 123456789,
 "Values":[{,"Id":0,"Type":0, "Value":true},
           {,"Id":1,"Type":0, "Value":true},
           {,"Id":8,"Type":29, "Value":2500000},
           {,"Id":15,"Type":29, "Value":10000000}]}
```

Response:

```
{"Status":"OK"}
```

### 4.2.4.6.2 Channel Configuration

JSON protocol accesses the IO channel configuration parameters by commands IoCfgSet (→ 5.3.2.4.3) and IoCfgGet (→ 5.3.2.4.3).

| | |
|---|---|
| "AO":{<br>"Mode": *mode*,<br>"Offset": *offset*<br>} | |

**Tab. 127 Analog Output Channel Configuration JSON Object**

| Name | Type | Description |
|---|---|---|
| Mode | Number | IO configuration parameter *outAnMode* (→ 4.2.4.3.1).<br>Parameter value Mode Byte in Tab. 125. |
| Offset | Number | IO configuration parameter *outAnOffset* (→ 4.2.4.3.2). |

**Tab. 128 Analog Output Channel Configuration JSON Object Values**

#### 4.2.4.6.2.1 Read Configuration

The JSON command IoCfgGet (→ 5.3.2.4.3) reads the IO channel configuration parameters of an IO channel.

Example
The device LIOT-AI8-10-AO8-10 has 8 analog 10V input channels (Channel numbers 0 to 7) and 8 analog 10V output channels (Channel numbers 8 to 15).

The configuration of the channel numbers 4 and 7 are read:

Request
```
{"Cmd": "IoCfgGet",
 "SessionId": 123456789,
 "ValueIds":[0,3]}
```

Response
```
{"Status": "OK",
 "Configs":[{
  "Id": 4,
  "Type": 3,
  "AO":{
   "Mode": 1,
   "Offset": 0 }},{
  "Id": 7,
  "Type": 3,
  "AO":{
   "Mode": 1,
   "Offset": -1000 }}]}
```

The response returns the IO configuration parameters of channel numbers 4 and 7. Both analog output channels are active. For IO channel 7 an offset of -1V (or -1mA) is configured.

#### 4.2.4.6.2.2 Update Configuration

The JSON command IoCfgSet (→ 5.3.2.4.3) writes the IO channel configuration parameters of an IO channel.

Example
The device LIOT-AI8-10-AO8-10 has 8 analog 10V input channels (Channel numbers 0 to 7) and 8 analog 10V output channels (Channel numbers 8 to 15).

IO channel 0 should be enabled, IO channel 7 should operate with an offset of -1V. All other parameters should remain unchanged.

Request
```
{"Cmd": "IoCfgSet",
 "SessionId": 123456789,
 "Configs":[{
  "Id": 4,
  "Type": 3,
  "AO":{
   "Mode": 1 }},{
  "Id": 7,
  "Type": 3,
```

```
"AO":{
 "Offset": -1000 }}]}
```

#### 4.2.4.7 MQTT Client

The IO channel values are accessible by the MQTT client (→ 3.3.10).

The IO channel values are selected by value identifiers (→ 3.3.10.4.1). They are formatted according to the value format settings (Tab. 129).

| Function Class Type | Value Formats |
|---|---|
| 5, 10, 24, 12S | Voltage (µV) Integer (Auto)<br>Voltage (V) Decimal<br>Voltage (V) Decimal Prec 0<br>Voltage (V) Decimal Prec 1<br>Voltage (V) Decimal Prec 2<br>Voltage (V) Decimal Prec 3 |
| 0M20, 4M20 | Current (nA) Integer (Auto)<br>Current (mA) Decimal<br>Current (mA) Decimal Prec 0<br>Current (mA) Decimal Prec 1<br>Current (mA) Decimal Prec 2<br>Current (mA) Decimal Prec 3 |

**Tab. 129 Analog Output Channel Value Formats**

#### 4.2.4.8 Modbus/TCP Server

The channel IO values are accessible by Modbus/TCP registers (→ 3.3.11.2).

#### 4.2.4.9 Data Logging

The analog output channel IO values can be logged (→ 3.4.3).

The IO channel values are selected by value identifiers (→ 3.3.10.4.1). They are formatted according to the value format settings (→ 3.3.10.4.4, Tab. 129).

## 4.2.5 RTD Input (RI8 Function Class)

The RI8 function class measures the resistance of 8 Pt100 or Pt1000 2-wire RTD sensors.

| Function Class | Value | Channels |
|---|---|---|
| RI8 | 0x0A10 | 8 |

**Tab. 130 RTD Sensor Input Function Classes**

| Function Class Type | Value | RTD Sensor Type | Input Signal Range | |
|---|---|---|---|---|
| | | | $T_{Min}$ | $T_{Max}$ |
| PT1000-180C | 0x1000 | Pt1000 | -180°C | 180°C |
| PT1000-0C360 | 0x1001 | Pt1000 | 0°C | 360°C |
| PT100-180C | 0x1010 | Pt100 | -180°C | 180°C |
| PT100-0C360 | 0x1011 | Pt100 | 0°C | 360°C |

**Tab. 131 RTD Sensor Input Function Class Types**

Tab. 130 and Tab. 131 list the RI8 function class and it's supported function class types with its' sensor types and measurement ranges.

### 4.2.5.1 IO Signal

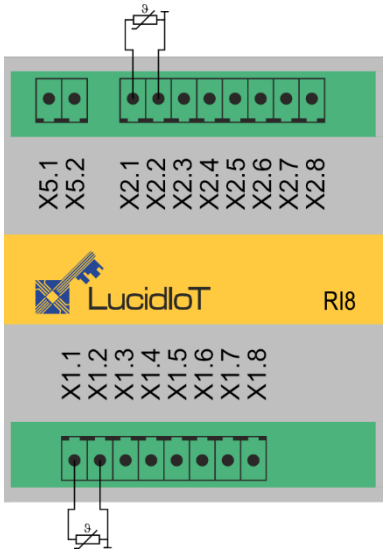



**Fig. 76 RI8 at IO Slot 0**

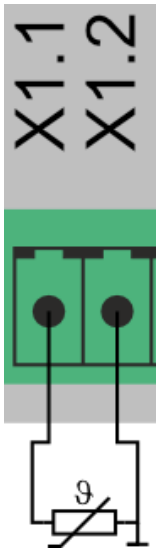



**Fig. 75 RI8 Signal**

Fig. 76 shows the connection section of the RI8 function installed at IO Slot 0.

A RTD sensor (Pt100 or Pt1000) is connected to X1.1 and X1.2 (RI0) and X2.1 and X2.2 (RI4).

Fig. 75 shows the RTD sensor connected to the IO terminals X1.1 and X1.2 (RI0) in detail.

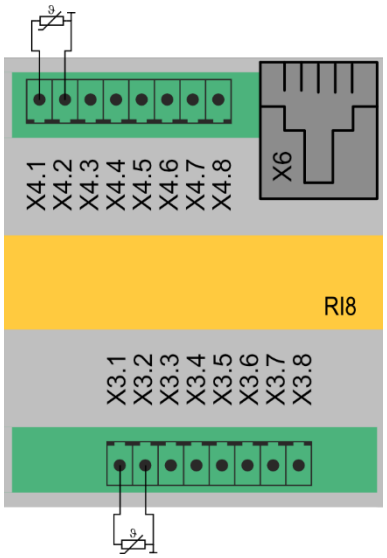



Fig. 77 shows the IO terminals of the RI8 function class installed at IO Slot 1.

A RTD sensor (Pt100 or Pt1000) is connected to X3.1 and X3.2 (RI0) and X4.1 and X4.2 (RI4).

**Fig. 77 RI8 at IO Slot 1**

lists the IO terminals of the RI8 function class.

| IO Terminal | IO Slot | Signal | IO Channel Number |
|---|---|---|---|
| X1.1 | 0 | RI0 | 0 |
| X1.2 | | GND | |
| X1.3 | | RI1 | 1 |
| X1.4 | | GND | |
| X1.5 | | RI2 | 2 |
| X1.6 | | GND | |
| X1.7 | | RI3 | 3 |
| X1.8 | | GND | |
| X2 | | RI4 ... RI7 | 4 – 7 |
| X3 | 1 | RI0 ... RI3 | 8 – 11 |
| X4 | | RI4 ... RI7 | 12 – 15 |

**Tab. 132 RI8 IO Terminal Connector**

### 4.2.5.2 Operation

The RI8 function class measures the resistance of 8 Pt100/Pt1000 RTD sensors.

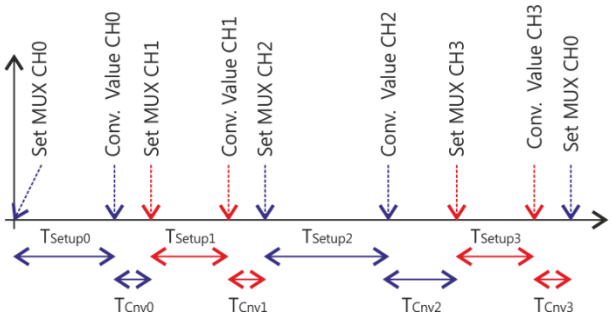



**Fig. 78 RTD Input Processing**

Fig. 78 shows the measurement process of the channel numbers 0 to 3.

Processing starts with activation of channel number 0. The RTD is sourced with a constant measurement current. The time $T_{Setup}$ allows the signal to settle. After it passed, the conversion starts. After the conversion

finished after $T_{Cnv}$ time, all other active channels are processed subsequently.

#### 4.2.5.2.1 Operation Modes

##### 4.2.5.2.1.1 Standard

In standard mode, the RTD input channels are acquired as explained.

##### 4.2.5.2.1.2 Inactive

In inactive mode, the RTD input channel processing is suspended. This improves the processing speed of other active channels.

Reading the IO value returns 0.

#### 4.2.5.2.2 Measurement Timing

Since all RTD sensors are acquired subsequently, the cycle time depends on $T_{Setup}$ and $T_{Cnv}$ of each active input channel. Inactive input channels are disabled improving the timing of active input channels.

$T_{Setup}$ is configured by IO configuration parameter *inRtSetupTime*.

#### 4.2.5.2.3 Offset Compensation

A measurement offset can be configured by IO configuration parameter *inRtOffset*.

The offset is configured in 0.1 Ω steps for Pt1000 sensors and 0.01 Ω steps for Pt100 sensors.

Example

The measured result of a 1000 Ω resistor connected to the input channel 0 of a Pt1000 function class should return the exact resistance value. If the value differs and e.g. the measurement wired adds additional 0.5 Ω this can be compensated.

```
LucidIoCtrl –dtcp:192.168.177.240:50001 –c0 –sinRtOffset=-5 –p
```

This command sets the offset of channel number 0 to -0.5 Ω.

#### 4.2.5.2.4 Line State Detection

The inputs channel can detect a broken or short-cut sensor line. The detection can be activated with *inRtTestOpen* (→ 4.2.5.3.2.1) and *inRtTestShort* (→ 4.2.5.3.2.2) parameters.

If the parameter *inRtTestOpen* is enabled and a broken line is detected, the IO value is set to ERR_OPEN.

If the parameter *inRtTestShort* is enabled and a shortcut is detected, the IO value is set to ERR_SHORT.

| Value Type | Condition | Returned value |
|---|---|---|
| TMS4 | ERR_SHORT | 0x80000000 |
| | ERR_OPEN | 0x7FFFFFFF |
| RMU4 | ERR_SHORT | 0 |
| | ERR_OPEN | 0xFFFFFFFF |

**Tab. 133 RTD Input Line State Errors**

#### 4.2.5.2.5 Oversampling and Averaging

The 12 Bit ADC accumulates 1024 samples and calculates resistance and temperature from it.

#### 4.2.5.2.6 Environment Temperature Compensation

The measurement of Pt100 sensor signals is more sensitive than Pt1000 sensor signals. Components of the measurement circuit (e.g. resistances) vary slightly over environment temperature, causing errors which are compensated by this function.

The environment temperature compensation is configured by the parameter *inRtTempComp* (→ 4.2.5.3.2.3). The parameter is only effective for Pt100 input channels.

### 4.2.5.3 Configuration Parameters

#### 4.2.5.3.1 inRtMode

The IO configuration parameter *inRtMode* specifies the operation mode (→ 4.2.5.2.1) of the RTD input channel.

| **Parameter** | *inRtMode* | **Access** | Read / Write |
|---|---|---|---|
| **Values** | Operation mode<br>inactive, standard | | |
| **Default Value** | standard | **Parameter Type** | - |

**Tab. 134 IO Channel Configuration Parameter inRtMode**

#### 4.2.5.3.2 inRtFlags

The IO configuration parameter *inRtFlags* specifies the operation flags of the RTD input channel.

##### 4.2.5.3.2.1 inRtTestOpen

The IO configuration parameter *inRtTestOpen* configures the returned value in case of an open (e.g broken) line is detected.

| **Parameter** | *inRtTestOpen* | **Access** | Read / Write |
|---|---|---|---|
| **Values** | on/off | | |
| **Default Value** | off | **Parameter Type** | 1 Bit |

**Tab. 135 IO Channel Configuration Parameter Bit inRtTestOpen**

#### 4.2.5.3.2.2 inRtTestShort

The IO configuration parameter *inRtTestShort* configures the returned value in case of a shortcut of the line is detected.

| **Parameter** | *inRtTestShort* | **Access** | Read / Write |
|---|---|---|---|
| **Values** | on/off | | |
| **Default Value** | off | **Parameter Type** | 1 Bit |

**Tab. 136 IO Channel Configuration Parameter Bit inRtTestShort**

#### 4.2.5.3.2.3 inRtTempComp

This IO configuration parameter configures the environment temperature compensation (→ 4.2.5.2.6).

| **Parameter** | *inRtTempComp* | **Access** | Read / Write |
|---|---|---|---|
| **Values** | on/off | | |
| **Default Value** | off | **Parameter Type** | 1 Bit |

**Tab. 137 IO Channel Configuration Parameter Bit inRtTempComp**

### 4.2.5.3.3 inRtOffset

The IO configuration parameter *inRtOffset* specifies an offset compensation value (→ 4.2.5.2.3).

| **Parameter** | *inRtOffset* | **Access** | Read / Write |
|---|---|---|---|
| **Values** | Pt1000<br>Offset Compensation in 0.1 Ω steps (-1,000 Ω ~ 1,000 Ω)<br>-10,000 ~ 10,000<br>Pt100<br>Offset Compensation in 0.01 Ω steps (-100 Ω ~ 100 Ω)<br>-10,000 ~ 10,000 | | |
| **Default Value** | 0 | | |

**Tab. 138 IO Channel Configuration Parameter inRtOffset**

### 4.2.5.3.4 inRtSetupTime

The IO configuration parameter *inRtSetupTime* specifies the setup time $T_{Setup}$

| **Parameter** | *inRtSetupTime* | **Access** | Read / Write |
|---|---|---|---|
| **Values** | $T_{Setup}$ in ms (milli seconds)<br>5 ms ≤ $T_{Setup}$ ≤ 1 s | | |
| **Default Value** | 25 (25 ms) | | |

**Tab. 139 IO Channel Configuration Parameter inRtSetupTime**

### 4.2.5.4 Web Interface

This section explains how the RI8 IO channel value and configuration are accessed using the web interface.

#### 4.2.5.4.1 IO Channel Value

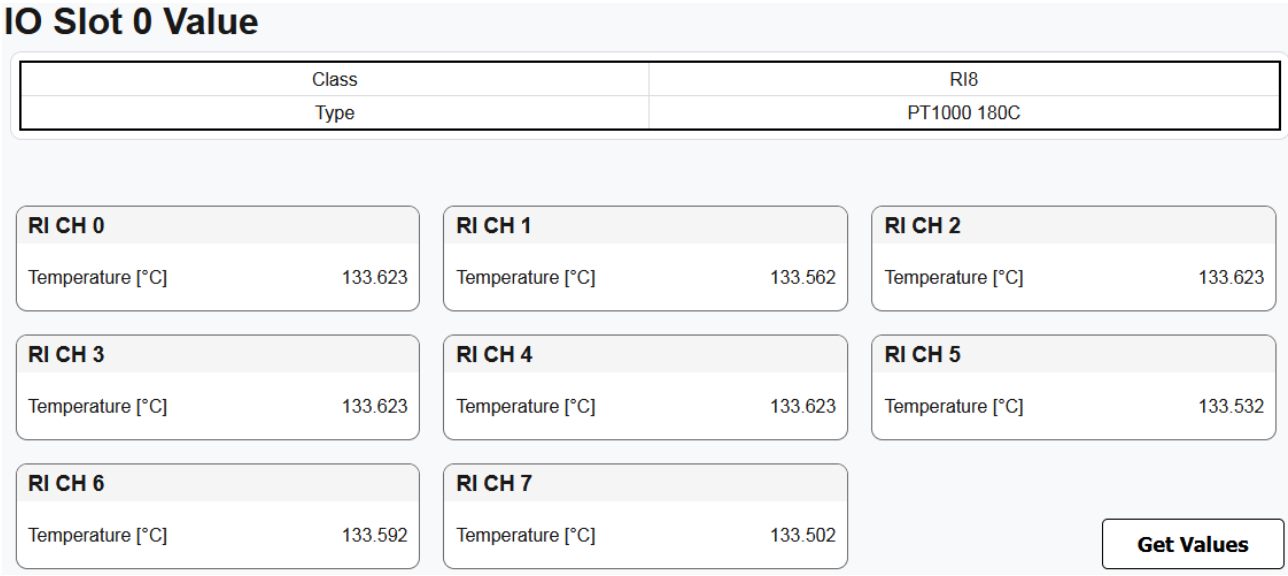



**Fig. 79 RTD Input Channel Value Access**

Fig. 79 shows the IO slot value page of the RTD input channels.

Only users with IO Value access rights are allowed to access this page. It is not visible for other users.

The IO values of all channels are read by clicking the *Get Values* button.

#### 4.2.5.4.2 IO Channel Configuration

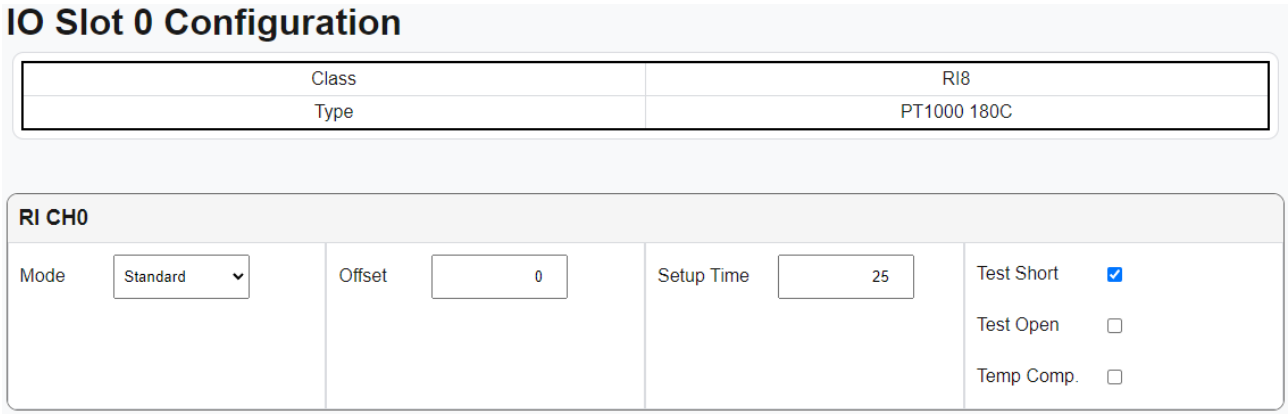



**Fig. 80 RTD Input Channel Configuration**

Fig. 80 shows the IO slot configuration page of the RTD input channel 0.

Only users with IO Config access rights are allowed to access this page. It is not visible for other users.

The IO Configuration parameters (→ 4.2.5.3) are updated and permanently saved by clicking the *Save* button.

### 4.2.5.5 FRAME Interface

This section explains how the RI8 IO channel value and configuration are accessed using the FRAME interface (→ 0) and the LucidIoCtrl command line tool.

#### 4.2.5.5.1 IO Channel Value

| Function Class | Supported FRAME Value Type |
|---|---|
| RI8 | TMS4, RSU2, RSU4 |

**Tab. 140 RTD Input IO Value Types**

The RI8 function class implements IO value types (→ 5.2) listed in Tab. 140.

##### 4.2.5.5.1.1 FRAME GetIo Command

The FRAME protocol command GetIo (→ 5.4.4.1) reads and returns the value of an IO channel number.

| **Command** | GetIo | **Access** | Read |
|---|---|---|---|
| **Opcode** | 0x46 | | |
| **LucidIoControl Command Line Tool** | | | |
| **Call (-tT)** | `LucidIoCtrl –dtcp:[ip:port] –c[Channel] –tT –r` | | |
| **Return** | `CHn:t`<br>n – Input Channel<br>t – Temperature in °C | | |
| **Call (-tR)** | `LucidIoCtrl –dtcp:[ip:port] –c[Channel] –tR –r` | | |
| **Return** | `CHn:r`<br>n – Input Channel<br>r – Resistance in Ω | | |

**Tab. 141 RTD Input FRAME GetIo Comand and LucidIoCtrl Arguments**

LucidIoCtrl Command Line Tool Example

Read temperature from input channel number 0.

```
      LucidIoCtrl –dtcp:192.168.177.240:50001 –c0 –tT –r
->    CH0:100.200
```

Read the corresponding resistance of the same input.

```
      LucidIoCtrl –dtcp:192.168.177.240:50001 –c0 –tR –r
->    CH0:1385.8
```

#### 4.2.5.5.1.2 FRAME GetIoGroup Command

The FRAME protocol command GetIoGroup (→ 5.4.4.2) reads and returns the values of a group of IO channel numbers.

| **Command** | GetIoGroup | **Access** | Read |
|---|---|---|---|
| **Opcode** | 0x48 | | |
| **LucidIoControl Command Line Tool** | | | |
| **Call (-tT)** | `LucidIoCtrl –dtcp:[ip:port] –c[Channels] –tT –r`<br><br>Channels:<br>Comma separated list of channels e.g. –c0,1,3 | | |
| **Return** | List of values sorted from lower to higher channels<br>`CHn:t`<br>n – Input Channel<br>t – Temperature in °C | | |
| **Call (-tR)** | `LucidIoCtrl –dtcp:[ip:port] –c[Channels] –tR –r`<br><br>Channels:<br>Comma separated list of channels e.g. –c0,1,3 | | |
| **Return** | `CHn:r`<br>n – Input Channel<br>r – Resistance in Ω | | |

**Tab. 142 RTD Input FRAME GetIoGroup Comand and LucidIoCtrl Arguments**

LucidIoCtrl Command Line Tool Example

Read temperatures from input channels numbers 0, 1, 2 and 7.

```
      LucidIoCtrl -dtcp:192.168.177.240:50001 -c0,1,2,7 -tT -r
->    CH0:100.000  CH1:0.500  CH2:-100.300  CH7:78.250
```

Read temperatures form input channel numbers 0, 1, 2 and 7.

```
      LucidIoCtrl -dtcp:192.168.177.240:50001 -c0,1,2,7 -tT -r
->    CH0:100.000  CH1:0.500  CH2:ERR_SHORT  CH7:ERR_OPEN
```

In this example, the RTD connected to input channel number 2 is shortcut and no RTD is connected to input channel number 7.

#### 4.2.5.5.2 IO Channel Configuration

IO channel configuration parameters are set by the FRAME protocol command SetParam (→ 5.4.4.5).

IO channel configuration parameters are read by the FRAME protocol command GetParam (→ 5.4.4.6).

##### 4.2.5.5.2.1 inRtMode

FRAME protocol access to *inRtMode* IO configuration parameter (→ 4.2.5.3.1).

<table>
<tr><td>Parameter</td><td>inRtMode</td><td>Address</td><td>0x1100</td></tr>
<tr><td>Values</td><td colspan="3">Input Mode<table><tr><th>Mode</th><th>Byte</th><th>Mode Value</th></tr><tr><td>Inactive</td><td>0x00</td><td>inactive</td></tr><tr><td>Standard</td><td>0x01</td><td>standard</td></tr></table></td></tr>
<tr><td></td><td></td><td>Parameter Type</td><td>1 Byte unsigned</td></tr>
<tr><td colspan="4">LucidIoCtrl Command Line Tool</td></tr>
<tr><td>Parameter Name</td><td>inRtMode</td><td>Parameter Values</td><td>Mode Value</td></tr>
<tr><td>Call (Set)</td><td colspan="3">LucidIoCtrl -dtcp:[ip:port] -c[Channel]<br>-sinRtMode=[Mode Value] {-p} {--default}</td></tr>
<tr><td>Call (Get)</td><td colspan="3">LucidIoCtrl -dtcp:[ip:port] -c[Channel] -ginRtMode</td></tr>
</table>

**Tab. 143 Configuration Parameter inRtMode (FRAME Protocol and LucidIoCtrl)**

LucidIoCtrl Command Line Tool Example

Set operation mode of input channel number 0 to standard mode and make the setting persistent.

```
      LucidIoCtrl -dtcp:192.168.177.240:50001 -c0 -sinRtMode=standard -p
```

Read the operation mode of input channel 0.

```
      LucidIoCtrl -dtcp:192.168.177.240:50001 -c0 -ginRtMode
      ->inRtMode=standard
```

##### 4.2.5.5.2.2 inRtFlags

FRAME protocol access to IO configuration parameter *inRtFlags* (→ 4.2.5.3.2).

| **Parameter** | *inRtFlags* | **Address** | 0x1101 |
|---|---|---|---|
| **Values** | Consists of the following Bit Parameters<br>**Bit Parameter** / **Bit Position**<br>*inRtTestOpen* / Bit 0<br>*inRtTestShort* / Bit 1<br>*inRtTempComp* / Bit 4 | | |
| **Default Value** | 0x00 | **Parameter Type** | 1 Byte unsigned |

**Tab. 144 IO Configuration Parameter inRtFlags (FRAME Protocol)**

The IO configuration parameter *inRtFlags* cannot be accessed by LucidIoCtrl directly. Use the bit parameters instead.

When *inRtFlags* is changed by the SetParam FRAME command (→ 5.4.4.5), it must be done in a read-modify-write sequence in order to prevent overwriting other parameter bits.

#### 4.2.5.5.2.2.1 inRtTestOpen

FRAME protocol access to IO configuration parameter *inRtTestOpen* (→ 4.2.5.3.2.1).

| **Parameter** | *inRtTestOpen* | **Address** | 0x1101 (Bit 0) |
|---|---|---|---|
| **Values** | on/off | **Parameter Type** | 1 Bit |
| **LucidIoCtrl Command Line Tool** | | | |
| **Parameter Name** | *inRtTestOpen* | **Parameter Values** | on/off |
| **Call (Set)** | `LucidIoCtrl -dtcp:[ip:port] -c[Channel] -sinRtTestOpen=[Value] {-p} {--default}` | | |
| **Call (Get)** | `LucidIoCtrl -dtcp:[ip:port] -c[Channel] -ginRtTestOpen` | | |

**Tab. 145 IO Configuration Parameter Bit inRtTestOpen (FRAME Protocol and LucidIoCtrl)**

#### 4.2.5.5.2.2.2 inRtTestShort

FRAME protocol access to IO configuration parameter *inRtTestShort* (→ 4.2.5.3.2.2).

| **Parameter** | *inRtTestShort* | **Address** | 0x1101 (Bit 1) |
|---|---|---|---|
| **Values** | on/off | **Parameter Type** | 1 Bit |
| **LucidIoCtrl Command Line Tool** | | | |
| **Parameter Name** | *inRtTestShort* | **Parameter Values** | on/off |
| **Call (Set)** | `LucidIoCtrl -dtcp:[ip:port] -c[Channel] -sinRtTestShort=[Value] {-p} {--default}` | | |
| **Call (Get)** | `LucidIoCtrl -dtcp:[ip:port] -c[Channel] -ginRtTestShort` | | |

**Tab. 146 IO Configuration Parameter Bit inRtTestShort (FRAME Protocol and LucidIoCtrl)**

#### 4.2.5.5.2.2.3 inRtTempComp

FRAME protocol access to IO configuration parameter *inRtTempComp* (→ 4.2.5.3.2.3).

| **Parameter** | *inRtTempComp* | **Address** | 0x1101 (Bit 4) |
|---|---|---|---|
| **Values** | on/off | **Parameter Type** | 1 Bit |
| **LucidIoCtrl Command Line Tool** | | | |
| **Parameter Name** | *inRtTempComp* | **Parameter Values** | on/off |
| **Call (Set)** | `LucidIoCtrl -dtcp:[ip:port] -c[Channel] -sinRtTempComp=[Value] {-p} {--default}` | | |
| **Call (Get)** | `LucidIoCtrl -dtcp:[ip:port] -c[Channel] -ginRtTempComp` | | |

**Tab. 147 IO Configuration Parameter Bit inRtTempComp (FRAME Protocol and LucidIoCtrl)**

#### 4.2.5.5.2.3 inRtOffset

FRAME protocol access to IO configuration parameter *inRtOffset* (→4.2.5.2.3).

| **Parameter** | *inRtOffset* | **Address** | 0x1120 |
|---|---|---|---|
| **LucidIoControl Command Line Tool** | | | |
| **Parameter Name** | *inRtOffset* | **Parameter Values** | Offset |
| **Call (Set)** | `LucidIoCtrl -dtcp:[ip:port] -c[Channel] -sinRtOffset=[Offset] {-p} {--default}` | | |
| **Call (Get)** | `LucidIoCtrl -dtcp:[ip:port] -c[Channel] -ginRtOffset` | | |

**Tab. 148 IO Configuration Parameter inRtOffset (FRAME Protocol and LucidIoCtrl)**

LucidIoCtrl Command Line Tool Example

Set offset compensation of the input channel number 0 (Pt1000) to -2Ω and make the setting persistent.

```
LucidIoCtrl -dtcp:192.168.177.240:50001 -c0 -sinRtOffset=-20 -p
```

Read Input Offset Compensation value

```
LucidIoCtrl -dtcp:192.168.177.240:50001 -c0 -ginRtOffset
-> inRtOffset=20
```

#### 4.2.5.5.2.4 inRtSetupTime

FRAME protocol access to IO configuration parameter *inRtSetupTime* (→ 4.2.5.3.4).

| **Parameter** | *inRtSetupTime* | **Address** | 0x1112 |
|---|---|---|---|
| **LucidIoCtrl Command Line Tool** | | | |
| **Parameter Name** | *inRtSetupTime* | **Parameter Values** | Time [ms] |
| **Call (Set)** | `LucidIoCtrl -dtcp:[ip:port] -c[Channel] -sinRtSetupTime=[Time] {-p} {--default}` | | |
| **Call (Get)** | `LucidIoCtrl -dtcp:[ip:port] -c[Channel] -ginRtSetupTime` | | |

**Tab. 149 IO Configuration Parameter inRtSetupTime (FRAME Protocol and LucidIoCtrl)**

LucidIoCtrl Command Line Tool Example

Set setup time $T_{Setup}$ of input channel number 0 to 25 ms and make the setting persistent.

```
LucidIoCtrl -dtcp:192.168.177.240:50001 -c0 -sinRtSetupTime=25 -p
```

Read $T_{Scan}$ parameter of input channel 0

```
LucidIoCtrl -dtcp:192.168.177.240:50001 -c0 -ginRtSetupTime
-> inRtSetupTime=25
```

#### 4.2.5.6 JSON Interface

This section explains how to access the IO channel value and configuration using the JSON interface (→ 3.3.14).

##### 4.2.5.6.1 IO Channel Value

###### 4.2.5.6.1.1 Read IO Channel Value

JSON protocol reads IO channel values by the command IoGet (→ 5.3.2.4.1).

Example:
Read IO channel values 0, 1, 8, and15 of a LIOT16-AO8-10V-RI8-PT1000 module with 8 analog 10V output channels and 8 Pt1000 RTD input channels.

Request:

```
{"Cmd":"IoGet",
 "SessionId: 123456789,
 "Values":[{"Id":0, "Type":29},
           {"Id":1, "Type":29},
           {"Id":8, "Type":65},
           {"Id":15, Type":65}]}
```

Response:

```
{"Status":"OK",
 "Values":[{"Id":0, "Type":29, "Value":2500000},
           {"Id":1, "Type":29, "Value":10000000},
           {"Id":8, Type":65, "Value":27000}
           {"Id":15, Type":65, "Value":123000}]}
```

The response returns for the analog output channels 0 and 1 2.5 V and 10V, for the RTD input channels 8 and 15 it returns 27°C and 123°C.

##### 4.2.5.6.2 IO Channel Configuration

JSON protocol accesses the IO channel configuration parameters by commands IoCfgSet (→ 5.3.2.4.3) and IoCfgGet (→ 5.3.2.4.3).

| | |
|---|---|
| "RI":{<br>"Mode": *modeByte*,<br>"TstOpn": testOpen,<br>"TstShrt": testShort,<br>"TCmp": tempComp<br>"SetupTime": *setupTime*,<br>"Offset": *offset*} | |

**Tab. 150 RTD Input Channel Configuration JSON Object**

| Name | Type | Description |
|---|---|---|
| Mode | Number | IO configuration parameter *inRtMode* (→ 4.2.5.3.1)<br>Parameter value Mode Byte in Tab. 143. |
| TstOpn | Boolean | IO configuration parameter *inRtTestOpen* (→ 4.2.5.3.2.1) |
| TstShrt | Boolean | IO configuration parameter *inRtTestShort* (→ 4.2.5.3.2.2) |

| | | |
|---|---|---|
| TCmp | Boolean | IO configuration parameter *inRtTempComp* (→ 4.2.5.3.2.3) |
| SetupTime | Number | IO configuration parameter *inRtSetupTime* (→ 4.2.5.3.4) |
| Offset | Number | IO configuration parameter *inRtOffset* (→ 4.2.5.3.3) |

**Tab. 151 RTD Input Channel Configuration JSON Object Values**

Fields of the RTD input channel configuration JSON object are optional.

#### 4.2.5.6.2.1 Read Configuration

The JSON command IoCfgGet (→ 5.3.2.4.3) reads the IO channel configuration parameters of an IO channel.

Example

The device LIOT-RI8-PT1000-180C-AO8-10V has 8 RTD input channels. The configuration of the channel numbers 0 and 7 are read:

Request

```
{"Cmd": "IoCfgGet",
 "SessionId": 123456789,
 "ValueIds":[0,7]}
```

Response

```
{"Status": "OK",
 "Configs":[{
  "Id": 0,
  "Type": 4,
  "RI":{
   "Mode": 1,
   "TCmp": false,
   "TstOpn": true,
   "TstShrt": true,
   "Offset": -10,
   "SetupTime": 25}},{
  "Id": 7,
  "Type": 4,
  "RI":{
   "Mode": 1,
   "TCmp": false,
   "TstOpn": true,
   "TstShrt": true,
   "Offset": -10,
   "SetupTime": 25}}]}
```

The response returns the IO configuration parameters of channel numbers 0 and 7. Both RTD input channels operate in standard mode with open and short detection enabled, a setup time of 25 ms and -1Ω offset correction.

#### 4.2.5.6.2.2 Update Configuration

The JSON command IoCfgSet (→ 5.3.2.4.3) writes the IO channel configuration parameters of an IO channel.

Example
The device LIOT-RI8-PT1000-180C-AO8-10V has 8 RTD input channels. The offset compensation of channel numbers 0 and 7 is set to -2 Ω:

Request
```
{"Cmd": "IoCfgSet",
 "SessionId": 123456789,
 "Configs":[{
  "Id": 0,
  "Type": 4,
  "RI":{
   "Offset": -20 }},{
  "Id": 7,
  "Type": 4,
  "RI":{
   "Offset": -20 }}]}
```

#### 4.2.5.7 MQTT Client

The IO channel values are accessible by the MQTT client (→ 3.3.10).

The IO channel values are selected by value identifiers (→ 3.3.10.4.1). They are formatted according to the value format settings (Tab. 152).

| Function Class Type | Value Formats |
|---|---|
| RI | Resistance (mΩ) Integer<br>Resistance (Ω) Decimal<br>Resistance (Ω) Decimal Prec 0<br>Resistance (Ω) Decimal Prec 1<br>Resistance (Ω) Decimal Prec 2<br>Resistance (Ω) Decimal Prec 3<br>Temperature (m°C) Integer (Auto)<br>Temperature (°C) Decimal<br>Temperature (°C) Decimal Prec 0<br>Temperature (°C) Decimal Prec 1<br>Temperature (°C) Decimal Prec 2<br>Temperature (°C) Decimal Prec 3 |

**Tab. 152 RTD Input Channel Value Formats**

#### 4.2.5.8 Modbus/TCP Server

The IO channel values are accessible by Modbus/TCP registers (→ 3.3.11.2).

32 bit IO value registers contain the temperature in 1/1000 °C, 16 bit IO value registers in 1/10 °C. The IO value registers are in signed format.

Resistance values are accessible by alternative IO value registers. 32 bit alternative IO value registers contain the resistance in mΩ, 16 bit registers in 1/10 Ω. The alternative IO value registers are in unsigned format.

#### 4.2.5.9 Data Logging

The RTD input channel IO values can be logged (→ 3.4.3).

The IO channel values are selected by value identifiers (→ 3.3.10.4.1). They are formatted according to the value format settings (→ 3.3.10.4.4, Tab. 152).

# 5 Data Exchange Protocol

This section describes the JSON objects and FRAME binary frame format in details.

The developer can take this documentation as reference interfacing the device e.g. when implementing a new customer specific API.

For standard use cases it is not necessary to read this section, but instead the documentation of the available API (e.g. C#, Python, Java, etc.).

The data exchange protocols operate in a request response principle. The host sends a request, LucidIoT executes it and responds.

## 5.1 Command Groups

Command groups contain commands with specific functions:

| Command Group | Commands in this Group |
|---|---|
| Config | Read and write of device configuration |
| IO Config | IO Channel configuration |
| IO Value | IO Channel value read and write operations |
| General | General commands |

**Tab. 153 Command Groups**

Most commands need authentication of the user. In order to execute them properly the user must at first log in by sending *AccLogin* command (→ 3.4.5.2, 5.3.2.1).

## 5.2 Value Types

The value type specifies the data format of IO values.

Supported value types are explained in Tab. 154. Support of value types depend on the IO channel functionality.

The column *Value Type* contains the abbreviation of the value type.

The column *Value Type Byte* identifies the value type for FRAME and JSON protocol.

The column *Frame Size* lists the number of bytes a value of this value type occupies in the FRAME format.

The column *JSON Type* contains the JSON standard type of which the value type is represented in JSON format.

| Value Type | Value Type Byte | Description | Frame Size | JSON Type |
|---|---|---|---|---|
| DI1 | 0x00 (0) | Digital Logic Value<br>Value for digital inputs and outputs.<br>Supported values are either “00” and “01”. | 1 Byte | Boolean |
| CNT2 | 0x0A (10) | Digital Counter Value<br>Input Counter for Digital Inputs.<br>Value Range: 0 ~ 65,535 | 2 Bytes | Number |
| VOS4 | 0x1D (29) | Voltage Value (signed)<br>Voltage for analog inputs and outputs<br>Resolution: 1 µV<br>Value Range: -100,000,000 µV ~ 100,000,000 µV | 4 Bytes | Number |
| CUS4 | 0x23 (35) | Current Value (signed)<br>Current for analog inputs and outputs<br>Resolution: 1 nA<br>Value Range: -1,000,000,000 nA ~ 1,000,000,000 nA | 4 Bytes | Number |
| TMS4 | 0x41 (65) | Temperature Value (signed)<br>Temperature for RTD inputs<br>Resolution: 1/100 K<br>Value Range: -1,000 °C ~ 1,000 °C<br>Values: -100,000 ~ 100,000 | 4 Bytes | Number |
| RSU2 | 0x50 (80) | Resistance Value<br>Resistance Value of RTD inputs<br>Resolution: 1/10 Ω<br>Value Range: 0 ~ 5,000 Ω | 2 Bytes | Number |
| RSU4 | 0x51 (81) | Resistance Value<br>Resistance Value of RTD inputs<br>Resolution: mΩ<br>Value Range: 0 ~ 5,000 Ω | 4 Bytes | Number |

**Tab. 154 Value Types**

## 5.3 JSON Protocol

The LucidIoT JSON protocol is a request-response based JSON compatible protocol, which gives full access to device.

The JSON protocol is used by the JSON interface.

The JSON protocol gives full access to LucidIoT. The web interface uses JSON protocol frames in the POST requests.

### 5.3.1 JSON Status Code

A response returned by JSON protocol contains a status code. It is returned in the JSON element *Status* as string.

| Status Codes | |
|---|---|
| OK | Request executed with success |
| NSUP | Command not supported, not implemented |
| INV_LEN | Invalid length<br>Data does not fit in buffer |
| INV_TOKEN | Invalid token |
| INV_ARG | Invalid argument |
| INV_VAL | Invalid value or value type<br>Passes value could not be converted |
| INV_ACC | Invalid access rights |
| ERR_INTERN | Intern error<br>Error is not further classified |
| ERR_EXIST | Resource exists<br>E.g. If user account could not be added because it already exists |
| ERR_RES | No resource available<br>E.g. all user accounts used |

**Tab. 155 JSON Protocol Status Code**

Tab. 155 lists the possible status codes.

If not otherwise stated, in case or error the status code is returned only.

### 5.3.2 Commands

This section describes a subset of the implemented commands, which are used in order to access IO values and IO value configuration.

The command is passed by JSON element *Cmd* as string.

#### 5.3.2.1 AccLogin

This command logs in a user and returns the identifier of the login session (SessionId) in case of success.

The *SessionId* identifies the logged in user. It can be passed to command requests in order to identify the user

JSON Protocol Request

```
{"Cmd": "AccLogin",
 "User": "username",
 "Pass": "password"}
```

**Tab. 156 JSON AccLogin Request**

| Name | Type | Description |
|---|---|---|
| User | String | User name, 4 ≤ length ≤ 8 characters |
| Pass | String | Password, 6 ≤ length ≤ 12 characters |

**Tab. 157 JSON AccLogin Request Values**

JSON Protocol Response (Success)

```
{"Status": "OK",
 "SessionId": "sessionId"}
```

**Tab. 158 JSON AccLogin Response**

| Name | Type | Description |
|---|---|---|
| SessionId | Number | Unique session identifier with 4 bytes (→ 3.4.5.2) |

**Tab. 159 JSON AccLogin Response Vaues**

#### 5.3.2.2 AccLogout

This command logs out a user.

JSON Protocol Request

```
{"Cmd": "AccLogout",
 "SessionId": "sessionid"}
```

**Tab. 160 JSON AccLogout Request**

| Name | Type | Description |
|---|---|---|
| SessionId | Number | Unique session identifier with 4 bytes |

**Tab. 161 JSON AccLogout Request Value**

JSON Protocol Response (Success)

```
{"Status": "OK"}
```

#### 5.3.2.3 AccLogCheck

Check if the passed session identifier is valid and a user account is logged in with it.
If Heartbeat is set, the login timer is reset, refreshing the login session.

JSON Protocol Request

```
{"Cmd": "AccLogCheck",
 "SessionId": "sessionid",
```

| | |
|---|---|
| `"Heartbeat": heartbeat}` | |

**Tab. 162 JSON AccLogCheck Request**

| Name | Type | Description |
|---|---|---|
| SessionId | Number | Session identifier of the user login to check |
| Heartbeat | Bool | If true the login timer is reset<br>If missing or false timer is not reset |

**Tab. 163 JSON AccLogout Request Values**

JSON Protocol Response

| | |
|---|---|
| `{"Status": "OK",`<br>`"User: "username"}` | |

**Tab. 164 JSON AccLogout Response**

| Name | Type | Description |
|---|---|---|
| User | String | User name logged in with session identifier |

**Tab. 165 JSON AccLogout Response Values**

#### 5.3.2.4 IO

This section documents the access to IO values and IO value configuration parameters using the JSON protocol.

The basic structure of the requests and responses is generic. The function class dependent information is stored in arrays and objects.

##### 5.3.2.4.1 IoGet

The JSON IoSet command reads the IO channel value(s) of one or more IO channel number(s).

JSON Protocol Generic Request

| | |
|---|---|
| `{"Cmd":"IoGet",`<br>`"SessionId": "sessionid",`<br>`"Values":[{...}, ..., {...}]}` | |

**Tab. 166 JSON IoGet Generic Request**

| Name | Type | Description |
|---|---|---|
| SessionId | Number | Session identifier of logged in user. |
| Values | Array | Array of value objects to read.<br>This array contains an array of up to 16 value objects. |

**Tab. 167 JSON IoSet Generic Request Values**

JSON Protocol Value Object (without Value)

| | |
|---|---|
| `{"Id": valueId,`<br>`"Type" : valueTypeByte}` | |

**Tab. 168 JSON Protocol Value Object (without IO Value)**

The protocol fields are explained in Tab. 177. For the IoGet command the protocol field *Value* is omitted.

JSON Protocol Response in Case of Success

```
{"Status":"OK",
 "Values":[{...}, .... {...}]}
```

**Tab. 169 JSON IoGet Response (Success)**

| Name | Type | Description |
|---|---|---|
| Values | Array | Array of returned value objects.<br>According to Tab. 177 with returned IO value in the *Value* field. |

**Tab. 170 JSON IoGet Response (Success) Values**

JSON Protocol Response for error

```
{"Status":"EXEC_ERR",
 "Id": valueId}         <- Value ID of first error
```

Example

Reading back IO values of the LIOT-DO8-I-AO8-10V device explained in 4.1.1.

Read back the digital output channel values of channel number 0 and 7.

Read back the analog output voltages of channel number 11.

Value type VOS4 = 0x1D -> Type = 29.

Session identifier is 123456789 in this example.

Request:

```
{"Cmd":"IoGet",
 "SessionId: 123456789,
 "Values":[{"Id":0, "Type":0 },
           {"Id":7, "Type":0 },
           {"Id":11, Type":29}]}
```

Response:

```
{"Status":"OK",
 "Values":[{"Id":0, "Type":0, "Value":true},
           {"Id":7, "Type":0, "Value":true},
           {"Id":11, Type":29, "Value":5.0}]}
```

The response returns for the two digital output channels IO value of "1", and 5.0V for the analog output channel.

### 5.3.2.4.2 IoSet

The JSON IoSet command writes the IO value(s) of one or more IO channel numbers.

JSON Protocol Generic Request

```
{"Cmd":"IoSet",
 "SessionId": "sessionid",
 "Save": save,
 "Values":[{...}, ..., {...}]}
```

**Tab. 171 JSON IoSet Generic Request**

| Name | Type | Description |
|---|---|---|
| SessionId | Number | Session identifier of logged in user. |
| Save | Boolean | If set, the values are permanently stored in NV memory and restored when the device is started.<br>Write operations to the NV must not be executed periodically in order to avoid wear out of NV memory.<br>If not set or omitted, the values are used temporarily until system restart. |
| Values | Array | Array of value objects to write.<br>This array contains an array of up to 16 value objects. |

**Tab. 172 JSON IoSet Generic Request Values**

JSON Protocol Response in Case of Success

```
{"Status":"OK"}
```

**Tab. 173 JSON IoSet Response (Success)**

JSON Protocol Response in Case of Error

```
{"Status":"EXEC_ERR",
 "Id": valueId}
```

**Tab. 174 JSON IoSet Response (Error)**

| Name | Type | Description |
|---|---|---|
| Id | Number | Value Identifier.<br>First value identifier, which caused error. |

**Tab. 175 JSON IoSet Response (Error) Values**

JSON Protocol Value Object (with Value)

```
{"Id": valueId,
 "Type" : valueTypeByte,
 "Value": ioValue}
```

**Tab. 176 JSON Protocol Value Object (with IO Value)**

| Name | Type | Description |
|---|---|---|
| Id | Number | Value Identifier.<br>When accessing IO channel values it contains the IO channel number. |
| Type | Number | Value Type Byte.<br>The value type of IO channel value.<br>(→ 5.2, Tab. 154) |
| Value | Note | IO Value<br>Note:<br>Type of the field is defined in Tab. 154. It depends on the value type. |

**Tab. 177 JSON Protocol Value Object Values**

Example

Writing the IO values of the LIOT-DO8-I-AO8-10V device explained in 4.1.1.
Set the digital output channel values of channel number 0 and 7 to "1".
Set the analog output voltages of channel number 11 to 5.0V.
Value type VOS4 = 0x1D -> Type = 29.
Session identifier is 123456789 in this example.

Request:

```
{"Cmd":"IoSet",
 "SessionId: 123456789,
 "Values":[{"Id":0, "Type":0, "Value":true},
           {"Id":7, "Type":0, "Value":true},
           {"Id":11, Type":29, "Value":5.0}]}
```

Response:

```
{"Status":"OK"}
```

#### 5.3.2.4.3 IoCfgGet

The JSON IoCfgGet command reads the IO channel configuration parameters of one or more IO channel numbers.

JSON Protocol Request

| | |
|---|---|
| {"Cmd": "IoCfgGet",<br>"SessionId": *sessionid*,<br>"ValueIds":[..., ...]} | |

**Tab. 178 JSON IoCfgGet Request**

| Name | Type | Description |
|---|---|---|
| SessionId | Number | Session identifier of logged in user. |
| ValueIds | Array | Array with value identifiers |

**Tab. 179 JSON IoCfgGet Request Values**

JSON Protocol Generic Response in case of Success

| | |
|---|---|
| {"Status":"OK",<br>"Configs":[{<br>"Id": *valueId*,<br>"Type": *configType*,<br>"DI" \| "DO" \| "AI" \| "AO" \| "RI":<br>{...}}]} | |

**Tab. 180 JSON IoCfgGet Generic Response**

| Name | Type | Description |
|---|---|---|
| Configs | Array | Array of IO channel configuration objects.<br>Configuration objects consist of an outer object and an inner, function class dependent object. (→ Tab. 186) |

**Tab. 181 JSON IoCfgGet Generic Response Values**

The inner configuration is explained in the function class dependent JSON IoCfgSet command section of this document (→ 4.2).

JSON Protocol Response in Case of Error

```
{"Status":"EXEC_ERR",
 "Id": valueId}
```

**Tab. 182 JSON IoCfgGet Response (Error)**

| Name | Type | Description |
|---|---|---|
| Id | Number | First value identifier, which caused error. |

**Tab. 183 JSON IoCfgGet Response Values (Error)**

### 5.3.2.4.4 IoCfgSet

The JSON IoCfgSet command updates the IO channel configuration parameters of one or more IO channel numbers.

JSON Protocol Generic Request

```
{"Cmd": "IoCfgSet",
 "SessionId": sessionid,
 "Save": save,
 "Configs":[{
  "Id": valueId,
  "Type": configType,
  "DI" | "DO" | "AI" | "AO" | "RI":
   {...}}]}
```

**Tab. 184 JSON IoCfgSet Generic Request**

| Name | Type | Description |
|---|---|---|
| SessionId | Number | Session identifier of logged in user. |
| Save | Boolean | If set, the configuration is permanently stored in NV memory and restored when the device is started. Write operations to the NV must not be done periodically in order to avoid wear out of NV memory.<br>If not set or omitted, the values are used temporarily until system restart. |
| Configs | Array | Array of IO channel configuration objects.<br>Configuration objects consist of an outer object and an inner, function class dependent object. |

**Tab. 185 JSON IoCfgSet Generic Request Values**

| Name | Type | Description |
|---|---|---|
| Id | Number | Configuration value identifier. |
| Type | Number | Type of the inner configuration object. |

| Inner Configuration Object | Type |
|---|---|
| DI | 0 |
| DO | 1 |
| AI | 2 |
| AO | 3 |
| RI | 4 |

**Tab. 186 IoCfgSet Outer Configuration Object**

The inner configuration is explained in the function class dependent JSON IoCfgSet command section of this document (→ 4.2).

IO Configuration Object for Analog Outputs

```
"AO":{"Value": value,
 "Mode": mode,
 "RefreshTime": refreshTime,
 "Offset": offSet,
 "CalLow": calLow,
 "CalHigh": calHigh
}
```

IO Configuration Object for Analog Inputs

```
"AI":{"Value": value,
 "Mode": mode,
 "NrSamples": nrOfSamples,
 "Offset": offSet,
 "CalLow": calLow,
 "CalHigh": calHigh
}
```

IO Configuration Object for Temperature (Resistance) Inputs

```
"RI":{"Value": value,
 "Mode": mode,
 "SetupTime": setupTime,
 "NrSamples": nrOfSamples,
 "Offset": offSet,
 "CalLow": calLow,
 "CalHigh": calHigh
}
```

JSON Protocol Response for error

```
{"Status":"EXEC_ERR",
 "Id": valueId}         <- Value ID of first error
```

#### 5.3.2.4.5 IoCfgSet (Default)

The JSON IoCfgSet command updates the IO channel configuration parameters of one or more IO channel numbers with default values.

JSON Protocol Request

```
{"Cmd": "IoCfgSet",
 "Save": save,
 "Default": default,
 "ValueIds":[..., ...]} <- Array of Value Ids
```

## 5.4 FRAME Protocol

The FRAME protocol is used by the FRAME interface (→ 3.3.15).

The FRAME protocol is byte-oriented and compatible with other products of the LucidControl family.

The LucidIoCtrl command line tool makes use of the FRAME interface. In the

All values in the FRAME protocol are in little-endian byte order.

### 5.4.1 Request Frame

A request frame (Tab. 187) is sent by a host computer. It transfers data to the device and initiates the communication.

| Header Field | | | | Data Field |
|---|---|---|---|---|
| OPC | P1 | P2 | LEN | Data Field |
| 1 Byte | 1 Byte | 1 Byte | 1 Byte | LEN Bytes |

**Tab. 187 Structure of Request Frame**

| Field | Description |
|---|---|
| OPC | Opcode of Request Command |
| P1, P2 | Command specific Parameters |
| LEN | Number of Bytes in the Data Field |
| Data Field | Data Field |

**Tab. 188 Request Frame**

Tab. 188 explains the elements of the Request Frame. The Header Field is mandatory, the Data Field is optional and only available if LEN > 0.

| Header Field | | | | Data Field |
|---|---|---|---|---|
| OPC | P1 | P2 | LEN | Data Field |
| 1 Byte | n Bytes | 1 Byte | 1 Byte | LEN Bytes |

**Tab. 189 Extended Request Frame**

Requests can use the Extended Request Frame (Tab. 189).

The parameter P1 can be extended with optional bytes by setting the MSB (most significant bit) of P1.

| Header Field | | | | | |
|---|---|---|---|---|---|
| OPC | P1 | P1A | P1B | P2 | LEN |
| 0x48 | 0x81 | 0x81 | 0x02 | 0x00 | 0x00 |

**Tab. 190 Example of Extended Request Frame**

In Tab. 190, the GetIoGroup command (0x48) reads digital channels 0, 7 and 15. P1 = 0x81 selects IO channel number 0 and the set MSB defines P1A being present. P1A = 0x81 selects IO channel number 7 and the set MSB defines P1B being present. P1B = 0x02 selects IO channel number 15.

## 5.4.2 Response Frame

The Response Frame (Tab. 191) is returned by the device to the host computer.

| Header Field | | Data Field |
|---|---|---|
| Status | LEN | Data Field |
| 1 Byte | 1 Byte | LEN Bytes |

**Tab. 191 Structure of Response Frame**

| Field | Description |
|---|---|
| Status | Frame Status Code |
| LEN | Number of Bytes in Data Field |
| Data Field | Data Field |

**Tab. 192 Response Frame**

Tab. 192 explains the elements of the Response Frame. The Header Field is mandatory, the Data Field is optional and only available if LEN > 0.

Depending on the command and the execution status, the device may return more or less information.

In the case of success, the Frame Status Code (→ 5.4.3) has a value of 0x00 and LEN indicates how many bytes the Data Field contains.

In the case of error the Frame Status Code not 0x00 and the Length field indicates with a value of 0x00 that the Data Field is absent. A detailed description of Error Status Codes can be found in section 5.4.3.

## 5.4.3 Frame Status Code

Tab. 193 shows the Frame Status Codes and the corresponding Status Message.

| Status Code | Status | Status Message |
|---|---|---|
| 0x00 | OK | Request executed with success |
| 0xA0 | NO_SUPPORT | Command not supported |
| 0xB0 | INV_LENGTH | Invalid data length |
| 0xB2 | INV_P1 | Invalid parameter P1 |
| 0xB4 | INV_P2 | Invalid parameter P2 |
| 0xB6 | INV_VALUE | Invalid value or value type |
| 0xB8 | INV_CHANNEL | Invalid IO channel number |
| 0xBA | INV_PARAM | Invalid parameter address |
| 0xC0 | INV_DATA | Invalid data in data field |
| 0xD0 | ERR_EXECUTION | Error during command execution |

**Tab. 193 Frame Status Codes**

## 5.4.4 Commands

This section describes the commands supported by the FRAME protocol.

### 5.4.4.1 GetIo

The GetIo command reads the IO channel value.

In the case that the value of an output is read, the GetIo command returns the last written value.

Request Frame

| OPC | P1 | P2 | LEN |
|---|---|---|---|
| 0x46 | Channel | Value Type | 0 |

**Tab. 194 GetIo Request Frame**

| Value | Description |
|---|---|
| Channel | IO Channel Number (→ 4.1.1) |
| Value Type | Value Type Byte (→ 5.2) |

**Tab. 195 GetIo Request**

Response Frame

In case of successful execution, the command returns the value of the specified IO channel number.

| Status | LEN | Data Field |
|---|---|---|
| Status | Size | Value |

**Tab. 196 GetIo Response Frame**

Size represents the length of the value according to the Value Type Byte.

In the case of an error, the command returns Frame Status Code (→ 5.4.3) only.

Example
Read the IO channel number 3 of an analog voltage input or output.

Request Frame

| OPC | P1 | P2 | LEN |
|---|---|---|---|
| 0x46 | 0x03 | 0x1D | 0 |

**Tab. 197 GetIo Request Example**

Response Frame

| Header Field | | Data Field | | | |
|---|---|---|---|---|---|
| Status | LEN | Value | | | |
| 0x00 | 0x04 | 0xC0 | 0xB4 | 0xB3 | 0xFF |

**Tab. 198 GetIo Response Example**

The command returns a signed value of hexadeximal 0xFFB3B4C0 = 5000000µV = 5V

#### 5.4.4.2 GetIoGroup

The GetIoGroup command reads a group of IO channel values of the same value type.

In the case that the value of an output is read, the GetIoGroup command returns the last written value.

Request Frame

| OPC | P1 | P2 | LEN |
|---|---|---|---|
| 0x48 | Channel Mask | Value Type | 0 |

**Tab. 199 GetIoGroup Request Frame**

<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>Channel Mask</td><td>Channel Bit Mask specifying the IO channel number(s)
<table>
<tr><th>Channel</th><th>Bit P1</th><th>Bit P1A</th><th>Bit P1B</th><th>Value</th></tr>
<tr><td>0</td><td>0</td><td>-</td><td>-</td><td>0x01</td></tr>
<tr><td>1</td><td>1</td><td>-</td><td>-</td><td>0x02</td></tr>
<tr><td>2</td><td>2</td><td>-</td><td>-</td><td>0x04</td></tr>
<tr><td>3</td><td>3</td><td>-</td><td>-</td><td>0x08</td></tr>
<tr><td>4</td><td>4</td><td>-</td><td>-</td><td>0x10</td></tr>
<tr><td>5</td><td>5</td><td>-</td><td>-</td><td>0x20</td></tr>
<tr><td>6</td><td>6</td><td>-</td><td>-</td><td>0x40</td></tr>
<tr><td>7</td><td>7</td><td>0</td><td>-</td><td>P1=0x80, P1A = 0x01</td></tr>
<tr><td>8</td><td>7</td><td>1</td><td>-</td><td>P1=0x80, P1A = 0x02</td></tr>
<tr><td>9</td><td>7</td><td>2</td><td>-</td><td>P1=0x80, P1A = 0x04</td></tr>
<tr><td>10</td><td>7</td><td>3</td><td>-</td><td>P1=0x80, P1A = 0x08</td></tr>
<tr><td>11</td><td>7</td><td>4</td><td>-</td><td>P1=0x80, P1A = 0x10</td></tr>
<tr><td>12</td><td>7</td><td>5</td><td>-</td><td>P1=0x80, P1A = 0x20</td></tr>
<tr><td>13</td><td>7</td><td>6</td><td>-</td><td>P1=0x80, P1A = 0x40</td></tr>
<tr><td>14</td><td>7</td><td>7</td><td>0</td><td>P1=0x80, P1A = 0x80,<br>P1B = 0x01</td></tr>
<tr><td>15</td><td>7</td><td>7</td><td>1</td><td>P1=0x80, P1A = 0x80,<br>P1B = 0x02</td></tr>
</table>
Values can be bitwise combined.<br><br>Examples<br>Accessing channel numbers:<br>0 and 3 Value P1 = 0x01 OR 0x08 = 0x09<br>1 and 2 Value P1 = 0x02 OR 0x04 = 0x06<br>1, 2 and 7 Value P1 = 0x02 OR 0x04 = 0x86<br>Value P1A = 0x01 (for channel 7)</td></tr>
<tr><td>Value Type</td><td>Value Type Byte (→ 5.2)</td></tr>
</table>

**Tab. 200 GetIoGroup Request**

Response Frame

In case of successful execution, the command returns the value(s) of the IO Channel Number(s) specified in the Channel Mask.

| Status | LEN | Data Field |
|---|---|---|
| Status | Size | Value(s) |

**Tab. 201 GetIo Response Frame**

Size represents the length of the value according to the Value Type Byte.

In the case of an error, the command returns Frame Status Code (→ 5.4.3) only.

Example
Read the input channels 0 and 3 of an analog input or output.

Request Frame

| OPC | P1 | P2 | LEN |
|---|---|---|---|
| 0x48 | 0x09 | 0x1D | 0 |

**Tab. 202 GetIoGroup Request Example**

Response Frame

| Header Field | | Data Field | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Status | LEN | Value Channel 0 | | | | Value Channel 3 | | | |
| 0x00 | 0x08 | 0xC0 | 0xB4 | 0xB3 | 0xFF | 0x00 | 0x40 | 0x4B | 0x4C |

**Tab. 203 GetIoGroup Response Example**

The input channel number 0 returns -5V input channel number 3 returns 5V.

### 5.4.4.3 SetIo

The SetIo command writes an IO channel value.

Request Frame

| OPC | P1 | P2 | LEN | Data |
|---|---|---|---|---|
| 0x40 | Channel | Value Type | Length | Value |

**Tab. 204 SetIo Request Frame**

| Value | Description |
|---|---|
| Channel | IO Channel Number (→ 4.1.1) |
| Value Type | Value Type Byte (→ 5.2) |
| Length | Size of the value (→ 5.2) |
| Value | IO Value to write |

**Tab. 205 SetIo Request**

Response Frame

| Status | Length |
|---|---|
| Status | 0 |

**Tab. 206 SetIo Response**

The command does not return a data field.

In the case of an error, the command returns Frame Status Code (→ 5.4.3) only.

Example
The IO channel number 1 is a digital output channel. Set it to ON

Request Frame

| OPC | P1 | P2 | LEN | Data |
|---|---|---|---|---|
| 0x40 | 0x01 | 0x00 | 0x01 | 0x01 |

**Tab. 207 SetIo Request Example**

Response Frame

| Status | Length |
|---|---|
| 0x00 | 0x00 |

**Tab. 208 SetIo Response Example**

#### 5.4.4.4 SetIoGroup

The SetIoGroup command writes a group of IO channel values of the same value type.

Request Frame

| OPC | P1 | P2 | LEN | Data |
|---|---|---|---|---|
| 0x42 | Channel Mask | Value Type | Length | Value(s) |

**Tab. 209 SetIoGroup Request Frame**

| Value | Description |
|---|---|
| Channel Mask | Channel Bit Mask specifying the IO channel number(s) (→ 5.4.4.2) |
| Value Type | Value Type Byte (→ 5.2) |
| Value(s) | One or more values to set |

**Tab. 210 SetIoGroup Request**

Response Frame

| Status | Length |
|---|---|
| Status | 0 |

**Tab. 211 SetIoGroup Response**

The command does not return a data field.

In the case of an error, the command returns Frame Status Code (→ 5.4.3) only.

Example

Set IO channel numbers 0 and 3 to 2.5 V and 5.0 V.

Request Frame

| OPC | P1 | P2 | LEN | Data Field | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x42 | 0x09 | 0x1D | 0x08 | Value of IO Channel Number 0 | | | | Value of IO Channel Number 3 | | | |
| | | | | 0x00 | 0xA0 | 0x25 | 0x26 | 0x00 | 0x40 | 0x4B | 0x4C |

**Tab. 212 SetIoGroup Request Example**

Response Frame

| Status | Length |
|---|---|
| 0x00 | 0x00 |

**Tab. 213 SetIoGroup Response Example**

#### 5.4.4.5 SetParam

The SetParam command sets the IO configuration parameter of an IO channel number.

Request Frame

| OPC | P1 | P2 | LEN | Data Field | |
|---|---|---|---|---|---|
| 0xA0 | Channel | Option | Length | P-Address | P-Value |

**Tab. 214 SetParam Request Frame**

| Value | Description |
|---|---|
| Channel | IO Channel Number (→ 4.1.1) |
| Option | Parameter Set Options<br>**Bit Position / Value / Description**<br>0 / 0x01 / Set Default<br>If bit is set the parameter value is set to its default value. Length must be set to 0 and Data Field must be omitted.<br>7 / 0x80 / Persistent Parameter<br>Parameter is saved and restored on module startup.<br>Values can be bitwise or combined |
| Length | Length of the Data Field containing parameter address and parameter value.<br>Must be >= 0x02 which is the length of the parameter address. |
| P-Address | IO Configuration Parameter Address<br>2 Byte product dependent address (→ 4.2 sub-sections). |
| P-Value | IO Configuration Parameter Value<br>Product dependent parameter value (→ 4.2 sub-sections). |

**Tab. 215 SetParam Request**

Response Frame

| Status | Length |
|---|---|
| Status | 0 |

**Tab. 216 SetParam Response**

The command does not return a data field.

In the case of an error, the command returns Frame Status Code (→ 5.4.3) only.

Example

Set the IO channel configuration parameter *outDiCycleTime* (Parameter Address = 0x1110) of IO channel number 0 to 750 ms and make it persistent. The parameter has a length of 4 bytes.

Request Frame

| OPC | P1 | P2 | LEN | Data Field | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0xA0 | 0 | 0x80 | 0x06 | P-Address | | P-Value | | | |
| | | | | 0x10 | 0x11 | 0xB0 | 0x71 | 0x0B | 0x00 |

**Tab. 217 SetParam Request Example**

Response Frame

| Status | Length |
|---|---|
| 0x00 | 0x00 |

**Tab. 218 SetParam Response Example**

### 5.4.4.6 GetParam

The GetParam command returns the IO configuration parameter of an IO channel number.

Request Frame

| OPC | P1 | P2 | LEN | Data |
|---|---|---|---|---|
| 0xA2 | Channel | 0x00 | 0x02 | P-Address |

**Tab. 219 GetParam Request Frame**

| Value | Description |
|---|---|
| Channel | IO Channel Number (→ 4.1.1) |
| P-Address | IO Configuration Parameter Address<br>2 Byte product dependent address (→ 4.2 sub-sections). |

**Tab. 220 GetParam Request**

Response Frame

In case of successful execution, the command returns the IO configuration value of the specified IO channel number.

| Status | LEN | Data Field |
|---|---|---|
| Status | Size | P-Value |

**Tab. 221 GetParam Response Frame**

| Value | Description |
|---|---|
| P-Value | IO Configuration Parameter Value<br>Product dependent parameter value (→ 4.2 sub-sections). |

**Tab. 222 GetParam Response**

In the case of an error, the command returns Frame Status Code (→ 5.4.3) only.

Example
Read the parameter *outDiCycleTime* (Parameter Address = 0x1110) of output channel 0.

Request Frame

| OPC | P1 | P2 | LEN | Data | |
|---|---|---|---|---|---|
| 0xA2 | 0x00 | 0x00 | 0x02 | P-Address | |
| | | | | 0x10 | 0x11 |

**Tab. 223 GetParam Request Example**

Response Frame

| Status | LEN | Data Field | | | |
|---|---|---|---|---|---|
| 0x00 | 0x04 | 0xB0 | 0x71 | 0x0B | 0x00 |

**Tab. 224 GetParam Response Example**

The command returns a value of 750 ms.

### 5.4.4.7 GetId

The GetId command returns system identification data of the device.

Request Frame

| OPC | P1 | P2 | LEN |
|---|---|---|---|
| 0xC0 | 0x00 | Option | 0 |

**Tab. 225 GetId Request Frame**

| Value | Description |
|---|---|
| Option | Get Identification Option<br>RFU. Value must be 0 |

**Tab. 226 GetId Request**

Response Frame
In case of successful execution, the command returns system identification data.

| Status | LEN | Data Field | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x00 | 21 | FW Rev. | HW Rev. | IO Slot 0 Class | IO Slot 0 Type | Device Serial No. | IO Slot 0 Rev. | Res | IO Slot 1 Class | IO Slot 1 Type | IO Slot 1 Rev. |

**Tab. 227 GetId Response Frame (Slot 0 and Slot1)**

| Status | LEN | Data Field | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0x00 | 16 | FW Rev. | HW Rev. | IO Slot 0 Class | IO Slot 0 Type | Device Serial No. | IO Slot 0 Rev. | Res |

**Tab. 228 GetId Response Frame (Slot 0 only)**

| Value | Description |
|---|---|
| FW Rev. | Firmware Revision<br>2 bytes value identifying the firmware revision |
| HW Rev. | Hardware Revision<br>1 byte value identifying the platform hardware revision |
| IO Slot 0/1 Class | IO slot 0/1 Function Class<br>2 bytes function class for IO slot 0 or 1 (→ 4.1) |
| IO Slot 0/1 Type | IO slot 0/1 Function Class Type<br>2 bytes function class type for IO slot 0 or 1 |
| IO Sot 0/1 Rev. | IO slot 0/1 Function Revision<br>1 byte with revision of the function class for IO slot 0 or 1 |
| Device Serial No. | Serial number of the device<br>4 bytes unique identifier of the device |
| Res | Reserved Values<br>4 bytes reserved for future use |

**Tab. 229 GetId Response**

# 6 Specification

## 6.1 Mechanical Specification

| Parameter | Value |
|---|---|
| Dimension l x w x h | 107.6 mm x 89.6 mm x 60.8 mm |
| Weight (in total) | 200 g |
| Protection Class (DIN 40050) | IP20 |
| Assembly | Rail-Mount (EN 50022, TS35) |

**Tab. 230 Mechanical Specification**

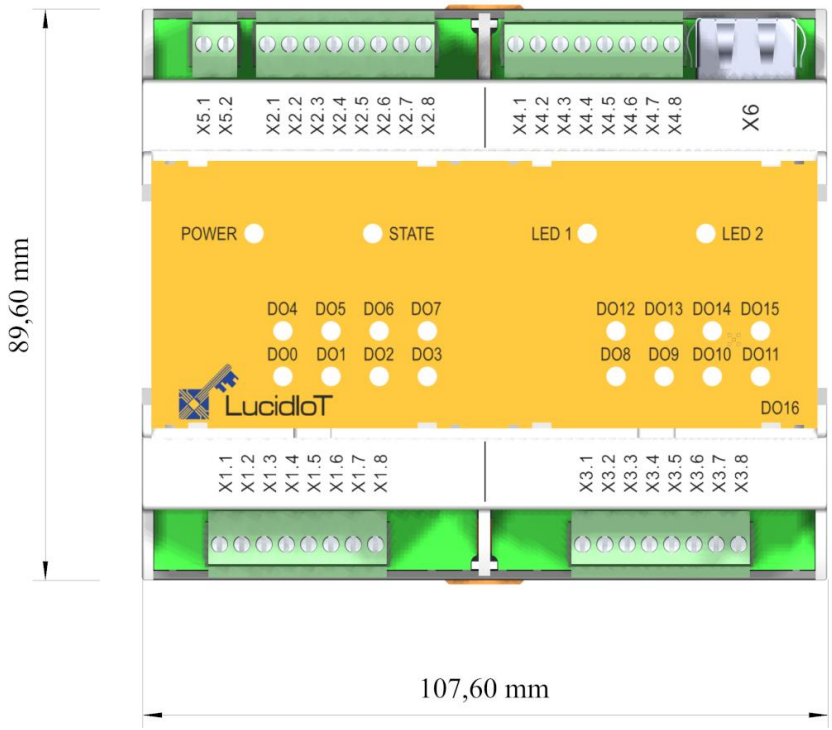



**Fig. 81 Top View**

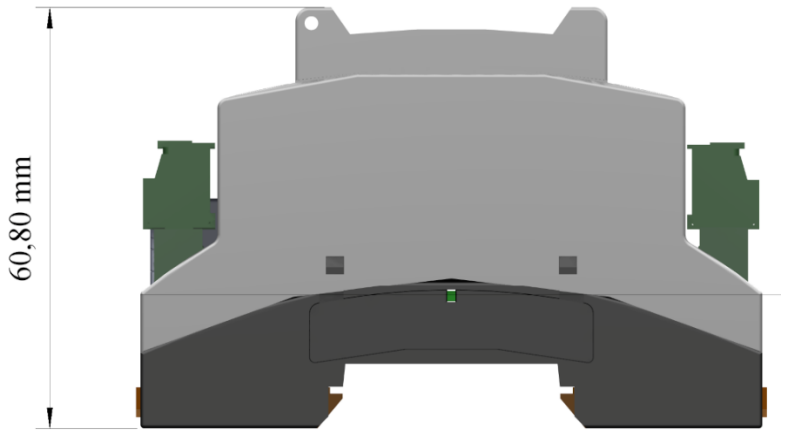



**Fig. 82 Side View**

## 6.2 General Specification

| Parameter | Value |
|---|---|
| **Electrical Characteristics** | |
| Power supply input voltage | 9 to 24 V |
| Power supply current rating | 1 A @ 12 V<br>0.5 A @ 24 V |
| **Environment** | |
| Temperature Storage | 0 °C ... +60 °C |
| Temperature Operation | 0 °C ... +55 °C |
| Humidity | < 85 % RH, non-condensing |

**Tab. 231 General Specification**

All electrical characteristics are specified at a room temperature of 25°C

## 6.3 DI8 - Digital Input Function Class

| Parameter | | Condition | Value |
|---|---|---|---|
| **Inputs** | | | |
| | No of Input Channels | | 8 |
| **Inputs - Electrical Characteristics** | | | |
| | Input Signal Maximum Low Level | 5V | $U_{05MaxLow}$ 2.5 V |
| | | 10V | $U_{10MaxLow}$ 6.0 V |
| | | 24V | $U_{24MaxLow}$ 16.0 V |
| | Input Signal Minimum High Level | 5V | $U_{05MinHigh}$ 3.5 V |
| | | 10V | $U_{10MinHigh}$ 8.5 V |
| | | 24V | $U_{24MinHigh}$ 21.0 V |
| | Maximum Input Voltage | | $U_{Max}$ 30 V |
| | Maximum Reverse Input Voltage | | $U_{RMax}$ -30 V |
| | Input Impedance | | $R_{In}$ > 1 kΩ |
| **Inputs – Timing Characteristic** | | | |
| | $T_{Scan}$ | | $t_{Min} < T_{Scan} < 1$ s |
| | $T_{Count}$ | | 1 ms < $T_{Count}$ < 1 h |
| | Minimum pulse length | | $t_{Min}$ 100 µs |
| | Maximum Frequency of Input Signal in Count Mode | DC 50 % | 2,000 Hz |
| | | 20% > DC < 80% | 500 Hz |

**Tab. 232 Specification of DI8 Function Class**

## 6.4 DO8 - Digital Output Function Class

| Parameter | | Condition | Value | |
|---|---|---|---|---|
| **Output Channels** | | | | |
| | No of Output Channels | | | 8 |
| **Outputs Channels - Electrical Characteristics** | | | | |
| | Maximum Rated Load Current Note1 | DO8-I | $I_{SSRMax}$ | 750 mA |
| | Maximum Rated Load Voltage | DO8-I | $U_{SSRMax}$ | 30 V |
| | Maximum On Resistance | DO8-I | $R_{SSR}$ | 0.25 Ω |
| **Outputs Channels – Timing Characteristic** | | | | |
| | Minimum On/Off Time $T_{OutDiMin}$ | DO8-I | $T_{OutDiMin}$ | 10 ms |
| | Timer resolution | | $T_{OutDiRes}$ | 1 µs |

**Tab. 233 Specification of DO8 Function Class**

Notes:

1) Output channels are able to control resistive loads only. For inductive loads additional protection is necessary.

## 6.5 AI8 - Analog Input Function Class

| Parameter | | Condition | Value | |
|---|---|---|---|---|
| **Input Channels** | | | | |
| | No of Input Channels | | | 8 |
| **Input Channels - Electrical Characteristics** | | | | |
| | Measurement Method | | | Analog to Digital Conversion |
| | Resolution | | | 12 bit |
| | Accuracy @25°C | | | typ. ±0.25 % of full-scale range |
| | Max. Measuring Error | | | ± 5 LSB |
| | Input Resistance | | $R_{In}$ | > 100 kΩ |
| **Input – Timing Characteristic** | | | | |
| | Acquisition Time / Channel | | $T_{Acq}$ Note1 | 50 ms |

**Tab. 234 Specification of AI8 Function Class**

Notes:

1) Typical value for 1 single channel acquisition with factor 1024 oversampling.
Typical value for 16 active channels 500ms for full cycle.
Typical value for 8 active channels 260ms for full cycle.
Faster acquisition on request.

Configurations with AI8 and RI8 will affect $T_{Acq}$. Please contact us for detailed information if acquisition time is important.

## 6.6 AO8 - Analog Output Function Class

| Parameter | | Condition | Value | |
|---|---|---|---|---|
| **Outputs** | | | | |
| | No of Output Channels | | | 8 |
| **Output - Electrical Characteristics** | | | | |
| | Output Function | | | Digital to Analog Conversion |
| | Resolution | | | 12 bit |
| | Accuracy | | | typ. ±0.25 % of full-scale range |
| | Max. Output Current | AO8-24 | $I_{TotalMax}$ | 40 mA |
| | | Others | | 160 mA |
| **Output – Electrical Characteristics of Current Outputs** | | | | |
| | Max. Output Voltage | | $U_{CChMax}$ | 10 V |
| | Min. Output Current [Note4] | AO8-4M20 | $I_{CChMin}$ | 4 mA |
| | | AO8-0M20 | | typ. 0.25 mA |
| | Output Current Dependence [Note3] | | $R_L$ not 500Ω | ±0.5 % |
| **Output – Electrical Characteristics of Voltage Outputs** | | | | |
| | Max. Output Current per Channel | AO8-24 | $I_{VChMax}$ | 10 mA [Note1] |
| | | Others | | 40 mA |
| | Min. Output Voltage [Note2] | AO8-24 | $U_{VChMin}$ | 0 V |
| | | Others | | 0 V |
| **Output – Timing Characteristic** | | | | |
| | Value Update interval | | $T_{Update}$ | typ. 10 ms |
| | Setup Time for stable output | | $T_{Stable}$ | typ. 1 ms |
| | DAC Conversion Time | | $T_{Conv}$ | typ. 1 ms |

**Tab. 235 Specification of AO8 Function Class**

Notes:

1) For currents > 2 mA an additional tolerance of max. 50 mV must be considered.
2) If not further specified, the modules are able to drive the minimum output value (0V) within the specified tolerances.
3) Outputs are calibrated to a base accuracy with 500Ω output resistors. For output resistances other than 500Ω, the Output Current Dependence applies.
4) For AO8-0M20, a minimum saturation current must be considered.

## 6.7 RI8 - RTD Input Function Class

| Parameter | | Condition | Value | |
|---|---|---|---|---|
| **Inputs** | | | | |
| | No of Input Channels | | | 8 |
| **Input - Electrical Characteristics** | | | | |
| | Measurement Method | | | RTD Two wire measurement |
| | Resolution | | | typ. 0.1°C |
| | RTD Type | | | Pt100 / Pt1000 DIN IEC 751 |
| | Measurement Error @25 °C | Pt100 | | typ. +/- 1.0°C |
| | | Pt1000 | | typ. +/- 0.5°C |
| | Constant Measurement Current | Pt100 | | 1mA |
| | | Pt1000 | | 0.5 mA |
| **Input – Timing Characteristic** | | | | |
| | Setup Time | | $T_{Setup}$ | 5 ms ≤ t ≤ 1 s |
| | Acquisition Time / Channel | | $T_{Acq}$ [1)] | typ. 50 ms |

**Tab. 236 Specification of RI8 Function Class**

Notes:

1) With default (25ms) setup time. Configurations with AI8 and RI8 will affect $T_{Acq}$. Please contact us for detailed information if acquisition time is important.

# 7 Ordering Information

This section lists all configurations of the LucidIoT.

Order Code Format:
LIOT-**C0**-**T0**-**C1**-**T1**

| Placeholder | |
|---|---|
| C0, C1 | Slot 0/1 function class (e.g. Analog Output 8 channels (AO8), Digital Input 8 channels (DI8) |
| T0, T1 | Slot 0/1 function class type (e.g. voltage range) |

**Tab. 237 Order Code Format**

Example
The order code LIOT-DI8-10V-AO8-10V refers to a device with 8 digital 10V input channels located at IO slot 0 and 8 analog 0-10V output channels located at IO slot 1.

List of Order Codes

| Order Code | IO Slot 0 | | IO Slot 1 | |
|---|---|---|---|---|
| | Function Class | Function Type | Function Class | Function Type |
| **16 CH Digital Inputs** | | | | |
| LIOT16-DI8-5V-DI8-5V | DI8 | 5V | DI8 | 5V |
| LIOT16-DI8-10V-DI8-5V | DI8 | 10V | DI8 | 5V |
| LIOT16-DI8-10V-DI8-10V | DI8 | 10V | DI8 | 10V |
| LIOT16-DI8-24V-DI8-5V | DI8 | 24V | DI8 | 5V |
| LIOT16-DI8-24V-DI8-10V | DI8 | 24V | DI8 | 10V |
| LIOT16-DI8-24V-DI8-24V | DI8 | 24V | DI8 | 24V |
| **Digital Input & Analog Input** | **8 CH Digital In** | | **8 CH Analog In** | |
| LIOT16-DI8-5V-AI8-5V | DI8 | 5V | AI8 | 5V |
| LIOT16-DI8-5V-AI8-10V | DI8 | 5V | AI8 | 10V |
| LIOT16-DI8-5V-AI8-10VS | DI8 | 5V | AI8 | ±10V |
| LIOT16-DI8-5V-AI8-24V | DI8 | 5V | AI8 | 24V |
| LIOT16-DI8-5V-AI8-20MA | DI8 | 5V | AI8 | 20mA |
| LIOT16-DI8-10V-AI8-5V | DI8 | 10V | AI8 | 5V |
| LIOT16-DI8-10V-AI8-10V | DI8 | 10V | AI8 | 10V |
| LIOT16-DI8-10V-AI8-10VS | DI8 | 10V | AI8 | ±10V |
| LIOT16-DI8-10V-AI8-24V | DI8 | 10V | AI8 | 24V |
| LIOT16-DI8-10V-AI8-20MA | DI8 | 10V | AI8 | 20mA |
| LIOT16-DI8-24V-AI8-5V | DI8 | 24V | AI8 | 5V |
| LIOT16-DI8-24V-AI8-10V | DI8 | 24V | AI8 | 10V |
| LIOT16-DI8-24V-AI8-10VS | DI8 | 24V | AI8 | ±10V |
| LIOT16-DI8-24V-AI8-24V | DI8 | 24V | AI8 | 24V |
| LIOT16-DI8-24V-AI8-20MA | DI8 | 24V | AI8 | 20mA |

| Digital Input & Analog Output | 8 CH Digital In | | 8 CH Analog Out | |
|---|---|---|---|---|
| LIOT16-DI8-5V-AO8-5V | DI8 | 5V | AO8 | 5V |
| LIOT16-DI8-5V-AO8-10V | DI8 | 5V | AO8 | 10V |
| LIOT16-DI8-5V-AO8-24V | DI8 | 5V | AO8 | 24V |
| LIOT16-DI8-5V-AO8-20MA | DI8 | 5V | AO8 | 20mA |
| LIOT16-DI8-5V-AO8-4MA20 | DI8 | 5V | AO8 | 4-20mA |
| LIOT16-DI8-10V-AO8-5V | DI8 | 10V | AO8 | 5V |
| LIOT16-DI8-10V-AO8-10V | DI8 | 10V | AO8 | 10V |
| LIOT16-DI8-10V-AO8-24V | DI8 | 10V | AO8 | 24V |
| LIOT16-DI8-10V-AO8-20MA | DI8 | 10V | AO8 | 20mA |
| LIOT16-DI8-10V-AO8-4MA20 | DI8 | 10V | AO8 | 4-20mA |
| LIOT16-DI8-24V-AO8-5V | DI8 | 24V | AO8 | 5V |
| LIOT16-DI8-24V-AO8-10V | DI8 | 24V | AO8 | 10V |
| LIOT16-DI8-24V-AO8-24V | DI8 | 24V | AO8 | 24V |
| LIOT16-DI8-24V-AO8-20MA | DI8 | 24V | AO8 | 20mA |
| LIOT16-DI8-24V-AO8-4MA20 | DI8 | 24V | AO8 | 4-20mA |
| **Digital Input & RTD Input** | **8 CH Digital In** | | **8 CH RTD In** | |
| LIOT16-DI8-5V-RI8-PT1000 | DI8 | 5V | RI8 | Pt1000 |
| LIOT16-DI8-5V-RI8-PT1000C360 | DI8 | 5V | RI8 | Pt1000 360°C |
| LIOT16-DI8-5V-RI8-PT100 | DI8 | 5V | RI8 | Pt100 |
| LIOT16-DI8-5V-RI8-PT100C360 | DI8 | 5V | RI8 | Pt100 360°C |
| LIOT16-DI8-10V-RI8-PT1000 | DI8 | 10V | RI8 | Pt1000 |
| LIOT16-DI8-10V-RI8-PT1000C360 | DI8 | 10V | RI8 | Pt1000 360°C |
| LIOT16-DI8-10V-RI8-PT100 | DI8 | 10V | RI8 | Pt100 |
| LIOT16-DI8-10V-RI8-PT100C360 | DI8 | 10V | RI8 | Pt100 360°C |
| LIOT16-DI8-24V-RI8-PT1000 | DI8 | 24V | RI8 | Pt1000 |
| LIOT16-DI8-24V-RI8-PT1000C360 | DI8 | 24V | RI8 | Pt1000 360°C |
| LIOT16-DI8-24V-RI8-PT100 | DI8 | 24V | RI8 | Pt100 |
| LIOT16-DI8-24V-RI8-PT100C360 | DI8 | 24V | RI8 | Pt100 360°C |
| **16 CH Digital Outputs** | | | | |
| LIOT16-DO8-I-DO8-I | DO8 | SSR | DO8 | SSR |
| **Digital Output & Digital Input** | **8 CH Digital Out** | | **8 CH Digital In** | |
| LIOT16-DO8-I-DI8-5V | DO8 | SSR | DI8 | 5V |
| LIOT16-DO8-I-DI8-10V | DO8 | SSR | DI8 | 10V |
| LIOT16-DO8-I-DI8-24V | DO8 | SSR | DI8 | 24V |
| **Digital Output & Analog Input** | **8 CH Digital Out** | | **8 CH Analog In** | |
| LIOT16-DO8-I-AI8-5V | DO8 | SSR | AI8 | 5V |
| LIOT16-DO8-I-AI8-10V | DO8 | SSR | AI8 | 10V |
| LIOT16-DO8-I-AI8-10VS | DO8 | SSR | AI8 | ±10V |
| LIOT16-DO8-I-AI8-24V | DO8 | SSR | AI8 | 24V |
| LIOT16-DO8-I-AI8-20MA | DO8 | SSR | AI8 | 20mA |

| Digital Output & Analog Output | 8 CH Digital Out | | 8 CH Analog Out | |
|---|---|---|---|---|
| LIOT16-DO8-I-AO8-5V | DO8 | SSR | AO8 | 5V |
| LIOT16-DO8-I-AO8-10V | DO8 | SSR | AO8 | 10V |
| LIOT16-DO8-I-AO8-12VS | DO8 | SSR | AO8 | ±12V |
| LIOT16-DO8-I-AO8-20MA | DO8 | SSR | AO8 | 20mA |
| LIOT16-DO8-I-AO8-4MA20 | DO8 | SSR | AO8 | 4-20mA |
| **Digital Output & RTD Input** | **8 CH Digital Out** | | **8 CH RTD In** | |
| LIOT16-DO8-I-RI8-PT1000 | DO8 | SSR | RI8 | Pt1000 |
| LIOT16-DO8-I-RI8-PT1000C360 | DO8 | SSR | RI8 | Pt1000 360°C |
| LIOT16-DO8-I-RI8-PT100 | DO8 | SSR | RI8 | Pt100 |
| LIOT16-DO8-I-RI8-PT100C360 | DO8 | SSR | RI8 | Pt100 360°C |
| **16 CH Analog Inputs** | | | | |
| LIOT16-AI8-5V-AI8-5V | AI8 | 5V | AI8 | 5V |
| LIOT16-AI8-10V-AI8-5V | AI8 | 10V | AI8 | 5V |
| LIOT16-AI8-10V-AI8-10V | AI8 | 10V | AI8 | 10V |
| LIOT16-AI8-10VS-AI8-5V | AI8 | ±10V | AI8 | 5V |
| LIOT16-AI8-10VS-AI8-10V | AI8 | ±10V | AI8 | 10V |
| LIOT16-AI8-10VS-AI8-10VS | AI8 | ±10V | AI8 | ±10V |
| LIOT16-AI8-24V-AI8-5V | AI8 | 24V | AI8 | 5V |
| LIOT16-AI8-24V-AI8-10V | AI8 | 24V | AI8 | 10V |
| LIOT16-AI8-24V-AI8-10VS | AI8 | 24V | AI8 | ±10V |
| LIOT16-AI8-24V-AI8-24V | AI8 | 24V | AI8 | 24V |
| LIOT16-AI8-20MA-AI8-5V | AI8 | 20MA | AI8 | 5V |
| LIOT16-AI8-20MA-AI8-10V | AI8 | 20MA | AI8 | 10V |
| LIOT16-AI8-20MA-AI8-10VS | AI8 | 20MA | AI8 | ±10V |
| LIOT16-AI8-20MA-AI8-24V | AI8 | 20MA | AI8 | 24V |
| LIOT16-AI8-20MA-AI8-20MA | AI8 | 20MA | AI8 | 20MA |
| **Analog Input & RTD Input** | **8 CH Analog In** | | **8 CH RTD In** | |
| LIOT16-AI8-5V-RI8-PT1000 | AI8 | 5V | RI8 | Pt1000 |
| LIOT16-AI8-5V-RI8-PT1000C360 | AI8 | 5V | RI8 | Pt1000 360°C |
| LIOT16-AI8-5V-RI8-PT100 | AI8 | 5V | RI8 | Pt100 |
| LIOT16-AI8-5V-RI8-PT100C360 | AI8 | 5V | RI8 | Pt100 360°C |
| LIOT16-AI8-10V-RI8-PT1000 | AI8 | 10V | RI8 | Pt1000 |
| LIOT16-AI8-10V-RI8-PT1000C360 | AI8 | 10V | RI8 | Pt1000 360°C |
| LIOT16-AI8-10V-RI8-PT100 | AI8 | 10V | RI8 | Pt100 |
| LIOT16-AI8-10V-RI8-PT100C360 | AI8 | 10V | RI8 | Pt100 360°C |
| LIOT16-AI8-10VS-RI8-PT1000 | AI8 | ±10V | RI8 | Pt1000 |
| LIOT16-AI8-10VS-RI8-PT1000C360 | AI8 | ±10V | RI8 | Pt1000 360°C |
| LIOT16-AI8-10VS-RI8-PT100 | AI8 | ±10V | RI8 | Pt100 |
| LIOT16-AI8-10VS-RI8-PT100C360 | AI8 | ±10V | RI8 | Pt100 360°C |

| | | | | |
|---|---|---|---|---|
| LIOT16-AI8-24V-RI8-PT1000 | AI8 | 24V | RI8 | Pt1000 |
| LIOT16-AI8-24V-RI8-PT1000C360 | AI8 | 24V | RI8 | Pt1000 360°C |
| LIOT16-AI8-24V-RI8-PT100 | AI8 | 24V | RI8 | Pt100 |
| LIOT16-AI8-24V-RI8-PT100C360 | AI8 | 24V | RI8 | Pt100 360°C |
| LIOT16-AI8-20MA-RI8-PT1000 | AI8 | 20MA | RI8 | Pt1000 |
| LIOT16-AI8-20MA-RI8-PT1000C360 | AI8 | 20MA | RI8 | Pt1000 360°C |
| LIOT16-AI8-20MA-RI8-PT100 | AI8 | 20MA | RI8 | Pt100 |
| LIOT16-AI8-20MA-RI8-PT100C360 | AI8 | 20MA | RI8 | Pt100 360°C |
| **16 CH Analog Outputs** | | | | |
| LIOT16-AO8-5V-AO8-5V | AO8 | 5V | AO8 | 5V |
| LIOT16-AO8-10V-AO8-5V | AO8 | 10V | AO8 | 5V |
| LIOT16-AO8-10V-AO8-10V | AO8 | 10V | AO8 | 10V |
| LIOT16-AO8-24V-AO8-5V | AO8 | 24V | AO8 | 5V |
| LIOT16-AO8-24V-AO8-10V | AO8 | 24V | AO8 | 10V |
| LIOT16-AO8-24V-AO8-24V | AO8 | 24V | AO8 | 24V |
| LIOT16-AO8-20MA-AO8-5V | AO8 | 20mA | AO8 | 5V |
| LIOT16-AO8-20MA-AO8-10V | AO8 | 20mA | AO8 | 10V |
| LIOT16-AO8-20MA-AO8-24V | AO8 | 20mA | AO8 | 24V |
| LIOT16-AO8-20MA-AO8-20MA | AO8 | 20mA | AO8 | 20mA |
| LIOT16-AO8-4MA20-AO8-5V | AO8 | 4-20mA | AO8 | 5V |
| LIOT16-AO8-4MA20-AO8-10V | AO8 | 4-20mA | AO8 | 10V |
| LIOT16-AO8-4MA20-AO8-24V | AO8 | 4-20mA | AO8 | 24V |
| LIOT16-AO8-4MA20-AO8-20MA | AO8 | 4-20mA | AO8 | 20mA |
| LIOT16-AO8-4MA20-AO8-4MA20 | AO8 | 4-20mA | AO8 | 4-20mA |
| **Analog Output & Analog Input** | **8 CH Analog Out** | | **8 CH Analog In** | |
| LIOT16-AO8-5V-AI8-5V | AO8 | 5V | AI8 | 5V |
| LIOT16-AO8-5V-AI8-10V | AO8 | 5V | AI8 | 10V |
| LIOT16-AO8-5V-AI8-10VS | AO8 | 5V | AI8 | ±10V |
| LIOT16-AO8-5V-AI8-24V | AO8 | 5V | AI8 | 24V |
| LIOT16-AO8-5V-AI8-20MA | AO8 | 5V | AI8 | 20mA |
| LIOT16-AO8-10V-AI8-5V | AO8 | 10V | AI8 | 5V |
| LIOT16-AO8-10V-AI8-10V | AO8 | 10V | AI8 | 10V |
| LIOT16-AO8-10V-AI8-10VS | AO8 | 10V | AI8 | ±10V |
| LIOT16-AO8-10V-AI8-24V | AO8 | 10V | AI8 | 24V |
| LIOT16-AO8-10V-AI8-20MA | AO8 | 10V | AI8 | 20mA |
| LIOT16-AO8-24V-AI8-5V | AO8 | 24V | AI8 | 5V |
| LIOT16-AO8-24V-AI8-10V | AO8 | 24V | AI8 | 10V |
| LIOT16-AO8-24V-AI8-10VS | AO8 | 24V | AI8 | ±10V |
| LIOT16-AO8-24V-AI8-24V | AO8 | 24V | AI8 | 24V |
| LIOT16-AO8-24V-AI8-20MA | AO8 | 24V | AI8 | 20mA |
| LIOT16-AO8-20MA-AI8-5V | AO8 | 20mA | AI8 | 5V |
| LIOT16-AO8-20MA-AI8-10V | AO8 | 20mA | AI8 | 10V |

| | | | | |
|---|---|---|---|---|
| LIOT16-AO8-20MA-AI8-10VS | AO8 | 20mA | AI8 | ±10V |
| LIOT16-AO8-20MA-AI8-24V | AO8 | 20mA | AI8 | 24V |
| LIOT16-AO8-20MA-AI8-20MA | AO8 | 20mA | AI8 | 20mA |
| LIOT16-AO8-4MA20-AI8-5V | AO8 | 4-20mA | AI8 | 5V |
| LIOT16-AO8-4MA20-AI8-10V | AO8 | 4-20mA | AI8 | 10V |
| LIOT16-AO8-4MA20-AI8-10VS | AO8 | 4-20mA | AI8 | ±10V |
| LIOT16-AO8-4MA20-AI8-24V | AO8 | 4-20mA | AI8 | 24V |
| LIOT16-AO8-4MA20-AI8-20MA | AO8 | 4-20mA | AI8 | 20mA |
| **Analog Output & RTD Input** | **8 CH Analog Out** | | **8 CH RTD In** | |
| LIOT16-AO8-5V-RI8-PT1000 | AO8 | 5V | RI8 | Pt1000 |
| LIOT16-AO8-5V-RI8-PT1000C360 | AO8 | 5V | RI8 | Pt1000 360°C |
| LIOT16-AO8-5V-RI8-PT100 | AO8 | 5V | RI8 | Pt100 |
| LIOT16-AO8-5V-RI8-PT100C360 | AO8 | 5V | RI8 | Pt100 360°C |
| LIOT16-AO8-10V-RI8-PT1000 | AO8 | 10V | RI8 | Pt1000 |
| LIOT16-AO8-10V-RI8-PT1000C360 | AO8 | 10V | RI8 | Pt1000 360°C |
| LIOT16-AO8-10V-RI8-PT100 | AO8 | 10V | RI8 | Pt100 |
| LIOT16-AO8-10V-RI8-PT100C360 | AO8 | 10V | RI8 | Pt100 360°C |
| LIOT16-AO8-24V-RI8-PT1000 | AO8 | 24V | RI8 | Pt1000 |
| LIOT16-AO8-24V-RI8-PT1000C360 | AO8 | 24V | RI8 | Pt1000 360°C |
| LIOT16-AO8-24V-RI8-PT100 | AO8 | 24V | RI8 | Pt100 |
| LIOT16-AO8-24V-RI8-PT100C360 | AO8 | 24V | RI8 | Pt100 360°C |
| LIOT16-AO8-20MA-RI8-PT1000 | AO8 | 20mA | RI8 | Pt1000 |
| LIOT16-AO8-20MA-RI8-PT1000C360 | AO8 | 20mA | RI8 | Pt1000 360°C |
| LIOT16-AO8-20MA-RI8-PT100 | AO8 | 20mA | RI8 | Pt100 |
| LIOT16-AO8-20MA-RI8-PT100C360 | AO8 | 20mA | RI8 | Pt100 360°C |
| LIOT16-AO8-4MA20-RI8-PT1000 | AO8 | 4-20mA | RI8 | Pt1000 |
| LIOT16-AO8-4MA20-RI8-PT1000C360 | AO8 | 4-20mA | RI8 | Pt1000 360°C |
| LIOT16-AO8-4MA20-RI8-PT100 | AO8 | 4-20mA | RI8 | Pt100 |
| LIOT16-AO8-4MA20-RI8-PT100C360 | AO8 | 4-20mA | RI8 | Pt100 360°C |
| **16 CH RTD Inputs** | | | | |
| LIOT16-RI8-PT1000-RI8-PT1000 | RI8 | Pt1000 | RI8 | Pt1000 |
| LIOT16-RI8-PT1000C360-RI8-PT1000 | RI8 | Pt1000 360°C | RI8 | Pt1000 |
| LIOT16-RI8-PT1000C360-RI8-PT1000C360 | RI8 | Pt1000 360°C | RI8 | Pt1000 360°C |
| LIOT16-RI8-PT100-RI8-PT1000 | RI8 | Pt100 | RI8 | Pt1000 |
| LIOT16-RI8-PT100-RI8-PT1000C360 | RI8 | Pt100 | RI8 | Pt1000 360°C |
| LIOT16-RI8-PT100-RI8-PT100 | RI8 | Pt100 | RI8 | Pt100 |
| LIOT16-RI8-PT100C360-RI8-PT1000 | RI8 | Pt100 360°C | RI8 | Pt1000 |

| | | | | |
|---|---|---|---|---|
| LIOT16-RI8-PT100C360-RI8-PT1000C360 | RI8 | Pt100 360°C | RI8 | Pt1000 360°C |
| LIOT16-RI8-PT100C360-RI8-PT100 | RI8 | Pt100 360°C | RI8 | Pt100 |
| LIOT16-RI8-PT100C360-RI8-PT100C360 | RI8 | Pt100 360°C | RI8 | Pt100 360°C |

**Tab. 238 List of Order Codes**

# 8 Document History

| Revision | Date | Comment |
|---|---|---|
| 0.9 | 2024/03/15 | Initial documentation |
| 0.9a | 2024/10/21 | Updated RI function class<br>• Open/Short line detection<br>• Environment temperature compensation<br>Updated picture in maintenance mode section |

**Tab. 239 Document History**