CAEN
Tools for Discovery

Electronic Instrumentation

PRELIMINARY

User Manual UM7615

N1081A Programmable Logic Unit

Software Development Kit

Rev. 2 - December 23rd, 2020

Purpose of this Guide

This Manual contains a brief description of the Software Development Kit for N1081A Four-Fold Programmable Logic Unit, including the basic knowledge and use examples of the Web Socket API for board configuration and data management.

Change Document Record

Date	Revision	Changes
June 22nd, 2020	00	Initial release
November 11th, 2020	01	Revised information in the input voltage threshold stage
December 23rd, 2020	02	Revised ordering options

Symbols, abbreviated terms and notation

API	Application Programming Interface
FPGA	Field Programmable Gate Array
OS	Operating system

Reference Document

[RD1] UM7385 – N1081A User Manual

CAEN S.p.A.
Via Vetraia, 11 55049 Viareggio (LU) - ITALY
Tel. +39.0584.388.398 Fax +39.0584.388.959
info@caen.it
www.caen.it

Index

List of Figures

List of Tables

1 Introduction

The **N1081A –Four Fould Programmable Logic Unit** is a laboratory tool that incorporates in a single **NIM** module the most common functionalities that you need to implement the **logic capabilities** of your experiment (see **[RD1]** for more details).

The board can be managed via remote control using a specific **Software Development Kit** (SDK) based on a **Web Socket API**. The SDK provides the needed commands to both configure the device and retrieve data from it.

A Python SDK is available for **free download** at https://github.com/NuclearInstruments/N1081A_SDK_Python

Figure 1.1: general view of the N1081A Programmable Logic Unit.

Models compatible with the described SDK are listed below:

Board	Description	Product Code
N1081A	N1081A – Four Fold Programmable Logic Unit	WN1081AXAAAA

Table 1.1: table of available models and accessories.

2 General Rules

The Web Socket protocol used in N1081A models provides a full-duplex communication channel over a single TCP connection. Most widely used web browsers support this protocol and provide a client to easily communicate with the Web Socket server on the device.

The URL that has to be used to open the web socket communication is: ***ws://<N1081A_ip>:8080/***

The N1081A_ip is shown on the instrument display home page.

All the request that can be sent to the device and all the messages received by the device are in a **JSON** format.

N1081A API support multiple users simultaneously connected to the instrument. During the developing, it is possible to keep the browser open on the N1081A Web Interface (see **[RD1]** for more details) in order to see the live effect of the performed operation.

There are two kind of requests that the user can send to the instrument: the commands to **set** the instrument configuration and the commands to **get** the instrument data. The instrument replies at each request.

The JSON messages to be sent to the instrument should always contain:

- the ***"command"*** field that specifies the type of request
- the ***"callback"*** field that can be defined by the user and is used to understand the origin of the message
- the parameters defined in the ***"params"*** field, which usually contains the ***"section"*** parameter that can assume the value 0, 1, 2 or 3 referring to the section A, B, C and D of the instrument.

The JSON messages received from the instrument as a reply, consist of:

- ***"Result"*** field which value can be True or False depending on the success of the communication
- ***"Response"*** describing the eventual error
- ***"missing command"***, ***"missing callback"*** and ***"missing parameters"*** indicate respectively that in the request the *"command"*, the *"callback"* or some parameters in the *"params"* field are missing.
- ***"invalid command"*** result means that the sent *"command"* is not valid
- The reported ***"callback"*** and ***"command"*** fields are the same of the request that has determined the instrument answer. If the sent request is a get command, the required information is enclosed in the *"data"* field.

3 Commands Description

This section describes the commands used to set and to get the function configuration of each section of the board and to retrieve the data generated by the defined functions.

Set Section

This command allows to assign a specific function to a determined device section.

```
{"command":"select_section_function","callback":"set_fn","params":{"section":0,"function
":"and"}
```

PARAMETERS	VALID VALUES	FUNCTION
section	0/1/2/3	Section A/B/C/D of the instrument
function	"wire", "and", "or", "or_veto", "veto", "majority", "majority_veto", "lut", "coincidence_gate", "scaler", "counter", "counter_timer", "chronom", "rate_meter", "rate_meter_advanced", "time_tag", "tof", "tot", "pulse_generator", "digital_generator", "pattern_generator"	Name of the function to be assigned to a section

Table 3.1: table of the Set Function configurable parameters.

Get Section

This command allows to get the name of the function assigned to each device section.

```
{"command":"get_all_sections_function","callback":"get_fn"}
```

The retrieved information is organized in the following way, with the “data” containing the function name of each section:

```
{"Response":"","Result":true,"callback":"get_fn","command":"get_all_sections_function","
data":[{"section":0,"function_name":"counter"},{"section":1,"function_name":"rate_meter_
advanced"},{"section":2,"function_name":"pulse_generator"},{"section":3,"function_name":
"digital_generator"}]}
```

Configure a function

Each function, in addition to some general settings, has specific parameters to be configured. Here are reported all the commands needed to configure all the functions. It is necessary to set the function of a section first, and then configure it using the correct configuration command.

Configure WIRE

This command allows to configure the WIRE function. The “lemo_enables” is an array of four elements, corresponding to the available input channels of the section, whose number should be in increasing order.

```
{"command":"configure_function","callback":"wire","params":{"section":0,"lemo_enables":[
{"lemo":0,"enable":true},{"lemo":1,"enable":true},{"lemo":2,"enable":true},{"lemo":3,"en
able":true}]}}
```

PARAMETERS	VALID VALUES	FUNCTION
section	0/1/2/3	Section A/B/C/D of the instrument
lemo	0/1/2/3	Input channel number
enable	true/false	Enable/Disable the correspondent input channel

Table 3.2: table of the WIRE function configurable parameters.

Configure AND

This command allows to configure the AND function. The "lemo_enables" is an array of six elements, corresponding to the available input channels of the section, whose number should be in increasing order.

```
{"command":"configure_function","callback":"and","params":{"section":0,"lemo_enables":[{
"lemo":0,"enable":true},{"lemo":1,"enable":true},{"lemo":2,"enable":true},{"lemo":3,"ena
ble":true},{"lemo":4,"enable":true},{"lemo":5,"enable":true}],"bypass_enable":false,"byp
ass_section":0}}
```

PARAMETERS	VALID VALUES	FUNCTION
section	0/1/2/3	Section A/B/C/D of the instrument
lemo	0/1/2/3/4/5	Input channel number
enable	true/false	Enable/disable the correspondent input channel
bypass_enable	true/false	Enable/disable the function bypass
bypass_section	0/1/2/3/4	Bypass section OFF/A/B/C/D (No the current section)

Table 3.3: table of the AND function configurable parameters.

Configure OR

This command allows to configure the OR function. The "lemo_enables" is an array of six elements, corresponding to the input channels of the section, whose number should be in increasing order.

```
{"command":"configure_function","callback":"or","params":{"section":0,"lemo_enables":[{"
lemo":0,"enable":true},{"lemo":1,"enable":true},{"lemo":2,"enable":true},{"lemo":3,"enab
le":true},{"lemo":4,"enable":true},{"lemo":5,"enable":true}],"bypass_enable":true,"bypas
s_section":2}}
```

PARAMETERS	VALID VALUES	FUNCTION
section	0/1/2/3	Section A/B/C/D of the instrument
lemo	0/1/2/3/4/5	Input channel number
enable	true/false	Enable/disable the correspondent input channel
bypass_enable	true/false	Enable/disable the function bypass
bypass_section	0/1/2/3/4	Bypass section OFF/A/B/C/D (No the current section)

Table 3.4: table of the OR function configurable parameters.

Configure OR+VETO

This command allows to configure the OR+VETO function. The "lemo_enables" is an array of five elements, corresponding to the input channels of the section, whose number should be in increasing order.

```
{"command":"configure_function","callback":"or_veto","params":{"section":0,"lemo_enables
":[{"lemo":0,"enable":true},{"lemo":1,"enable":true},{"lemo":2,"enable":true},{"lemo":3,
"enable":true},{"lemo":4,"enable":true}],"bypass_enable":false,"bypass_section":0}}
```

PARAMETERS	VALID VALUES	FUNCTION
section	0/1/2/3	Section A/B/C/D of the instrument
lemo	0/1/2/3/4	Input channel number
enable	true/false	Enable/disable the correspondent input channel
bypass_enable	true/false	Enable/disable the function bypass
bypass_section	0/1/2/3/4	Bypass section OFF/A/B/C/D (No the current section)

Table 3.5: table of the OR+VETO function configurable parameters.

Configure VETO

This command allows to configure the VETO function. The "lemo_enables" is an array of four elements, corresponding to the input channels of the section, whose number should be in increasing order.

{"command":"configure_function","callback":"veto","params":{"section":0,"lemo_enables":[{"lemo":0,"enable":true},{"lemo":1,"enable":true},{"lemo":2,"enable":true},{"lemo":3,"enable":true}]}}

PARAMETERS	VALID VALUES	FUNCTION
section	0/1/2/3	Section A/B/C/D of the instrument
lemo	0/1/2/3	Input channel number
enable	true/false	Enable/disable the correspondent input channel

Table 3.6: table of the VETO function configurable parameters.

Configure MAJORITY

This command allows to configure the MAJORITY function. The "lemo_enables" is an array of six elements, corresponding to the input channels of the section, whose number should be in increasing order.

{"command":"configure_function","callback":"majority","params":{"section":0,"lemo_enables":[{"lemo":0,"enable":true},{"lemo":1,"enable":true},{"lemo":2,"enable":true},{"lemo":3,"enable":true},{"lemo":4,"enable":true},{"lemo":5,"enable":true}]}}

PARAMETERS	VALID VALUES	FUNCTION
section	0/1/2/3	Section A/B/C/D of the instrument
lemo	0/1/2/3/4/5	Input channel number
enable	true/false	Enable/disable the correspondent input channel

Table 3.7: table of the MAJORITY function configurable parameters.

Configure MAJORITY+VETO

This command allows to configure the MAJORITY+VETO function. The "lemo_enables" is an array of five elements, corresponding to the input channels of the section, whose number should be in increasing order.

{"command":"configure_function","callback":"majority","params":{"section":0,"lemo_enables":[{"lemo":0,"enable":true},{"lemo":1,"enable":true},{"lemo":2,"enable":true},{"lemo":3,"enable":true},{"lemo":4,"enable":true},{"lemo":5,"enable":true}]}}

PARAMETERS	VALID VALUES	FUNCTION
section	0/1/2/3	Section A/B/C/D of the instrument
lemo	0/1/2/3/4	Input channel number
enable	true/false	Enable/disable the correspondent input channel

Table 3.8: table of the MAJORITY+VETO function configurable parameters.

Configure LUT

The LOOKUP TABLE (LUT) function requires a file to define all the combinations of the values assumed by each input and output channel. It is possible to configure the function by passing the content of the file to be stored and used or by specifying the file name if it is already stored into the device. The "lemo_in_enables" is an array of 6 elements, corresponding to the input channels, whose number should be in increasing order. The "lemo_out_enables" is an array of 4 elements, corresponding to the output channels, whose number should be in increasing order.

{"command":"configure_function","callback":"lut","params":{"section":0,"lemo_out_enables":[{"lemo":0,"enable":true},{"lemo":1,"enable":true},{"lemo":2,"enable":true},{"lemo":3,"enable":true}],"lemo_in_enables":[{"lemo":0,"enable":true},{"lemo":1,"enable":true},{"lemo":2,"enable":true},{"lemo":3,"enable":true},{"lemo":4,"enable":true},{"lemo":5,"enable":true}],"file_mode":1,"file_name":"lut","lut_values":[{"input":63,"output":0},{"input":0,"output":15},{"input":21,"output":0},{"input":63,"output":0}],"total_number":4}}

{"command":"configure_function","callback":"lut","params":{"section":0,"lemo_out_enables":[{"lemo":0,"enable":true},{"lemo":1,"enable":true},{"lemo":2,"enable":true},{"lemo":3,"enable":true}],"lemo_in_enables":[{"lemo":0,"enable":true},{"lemo":1,"enable":true},{"lemo":2,"enable":true},{"lemo":3,"enable":true},{"lemo":4,"enable":true},{"lemo":5,"enable":true}],"file_mode":0,"file_name":"lut"}}

PARAMETERS	VALID VALUES	FUNCTION
section	0/1/2/3	Section A/B/C/D of the instrument
lemo	0/1/2/3/4/5	Input or output channel number
enable	true/false	Enable/disable the correspondent channel
file_mode	0/1	Use an existing/create a new LUT file

file_name	At maximum 20 characters. Characters allowed: letters, numbers, _, + and -	LUT file name to be used (existing file or file to be created)
lut_values	[{},…,{}]	Content of the LUT file to be created: array of elements containing an input and an output value describing the logic state of the input and output channels with a binary value expressed as a decimal
total_number	Integer	Number of combinations in the LUT file to be created

Table 3.9: table of the LUT function configurable parameters.

Configure COINCIDENCE GATE

This command allows to configure the COINCIDENCE GATE function. The "lemo_enables" is an array of five elements, corresponding to the input channels of the section, whose number should be in increasing order.

```
{"command":"configure_function","callback":"coinc","params":{"section":0,"lemo_enables":
[{"lemo":0,"enable":true,"coincidence":true},{"lemo":1,"enable":true,"coincidence":true}
,{"lemo":2,"enable":true,"coincidence":true},{"lemo":3,"enable":true,"coincidence":true}
,{"lemo":4,"enable":true,"coincidence":true}],"gate":true,"close_on_coincidence":true,"d
elay":0,"width":300,"trigger":0}}
```

PARAMETERS	VALID VALUES	FUNCTION
section	0/1/2/3	Section A/B/C/D of the instrument
lemo	0/1/2/3/4	Input channel number
enable	true/false	Enable/disable the correspondent input channel
coincidence	true/false	Enable coincidence/anticoincidence mode
gate	true/false	Enable/disable external gate
close_on_coincidence	true/false	Enable/disable closing of the coincidence gate when a coincidence occurs
delay	0…100000	Delay of the coincidence gate in ns
width	0…100000	Width of the coincidence gate in ns
trigger	0/1/2/3/4/5	Coincidence gate opening mode: first arriving signal/ signal of the correspondent input channel

Table 3.10: table of the COINCIDENCE GATE function configurable parameters.

Configure SCALER

This command allows to configure the SCALER function. The "lemo_enables" is an array of four elements, corresponding to the input channels of the section, whose number should be in increasing order.

```
{"command":"configure_function","callback":"scaler","params":{"section":0,"scale":1,"lem
o_enables":[{"lemo":0,"enable":true},{"lemo":1,"enable":true},{"lemo":2,"enable":true},{
"lemo":3,"enable":true}],"gate":false}}
```

PARAMETERS	VALID VALUES	FUNCTION
section	0/1/2/3	Section A/B/C/D of the instrument
lemo	0/1/2/3	Input channel number
enable	true/false	Enable/disable the correspondent input channel
scale	1…100000000	Frequency divider value
gate	true/false	Enable/disable external gate

Table 3.11: table of the SCALER function configurable parameters.

Configure COUNTER

This command allows to configure the COUNTER function. The "lemo_enables" is an array of four elements, corresponding to the input channels of the section, whose number should be in increasing order.

```
{"command":"configure_function","callback":"counter","params":{"section":0,"lemo_enables
":[{"lemo":0,"enable":true},{"lemo":1,"enable":true},{"lemo":2,"enable":true},{"lemo":3,
"enable":true}],"gate":false}}
```

PARAMETERS	VALID VALUES	FUNCTION
section	0/1/2/3	Section A/B/C/D of the instrument
lemo	0/1/2/3	Input channel number
enable	true/false	Enable/disable the correspondent input channel
gate	true/false	Enable/disable external gate

Table 3.12: table of the COUNTER function configurable parameters.

Configure COUNTER TIMER

This command allows to configure the COUNTER TIMER function. The “lemo_enables” is an array of two elements, corresponding to the input channels of the section, whose number should be in increasing order. For the gate window and the target values it is possible to set a 64 bits number by separating the value in two 32 bits numbers.

```
{"command":"configure_function","callback":"counter_timer","params":{"section":0,"lemo_enables":[{"lemo":0,"enable":true},{"lemo":1,"enable":true}],"gate":false,"auto_reset":false,"gate_width1":0,"gate_width2":0,"source":0,"time":0,"mode":0,"target1":0,"target2":0}}
```

PARAMETERS	VALID VALUES	FUNCTION
section	0/1/2/3	Section A/B/C/D of the instrument
lemo	0/1	Input channel number
enable	true/false	Enable/disable the correspondent input channel
gate	true/false	Enable/disable external gate
auto_reset	true/false	Enable/disable the autoreset of the counts when the target is reached or the window is closed
gate_width1	0…2^32	32 least significant bits of the window value expressed in decimal notation
gate_width2	0…2^32	32 most significant bits of the window value expressed in decimal notation
source	0/1	Input channel/Internal timing as counter source
time	0/1/2/3	Time value in case of internal timing counter source: 10 ns/1 us/1 ms/1 s
mode	0/1/2/3	Counting mode: FREE/COUNTDOWN/TARGET/WINDOW
target1	0…2^32	32 least significant bits of the target value expressed in decimal notation
target2	0…2^32	32 most significant bits of the target value expressed in decimal notation

Table 3.13: table of the COUNTER TIMER function configurable parameters.

Configure CHRONOMETER

This command allows to configure the CHRONOMETER function. The “lemo_enables” is an array of two elements, corresponding to the input channels of the section, whose number should be in increasing order.

```
{"command":"configure_function","callback":"chronometer","params":{"section":0,"frequency":1,"mode":0,"reset_gate":false,"reset_stop":false,"lemo_enables":[{"lemo":0,"enable":true},{"lemo":1,"enable":true}],"gate":false}}
```

PARAMETERS	VALID VALUES	FUNCTION
section	0/1/2/3	Section A/B/C/D of the instrument
lemo	0/1	Input channel number
enable	true/false	Enable/disable the correspondent input channel
gate	true/false	Enable/disable external gate
frequency	1…100000000	Output signal frequency in Hz
mode	0/1	Chronometer mode: GATE/START-STOP
reset_gate	true/false	Enable/disable chronometer reset on gate closing in GATE mode
reset_stop	true/false	Enable/disable chronometer reset on stop signal in START-STOP mode

Table 3.14: table of the CHRONOMETER function configurable parameters.

Configure RATEMETER

This command allows to configure the RATEMETER function. The “lemo_enables” is an array of four elements, corresponding to the input channels of the section, whose number should be in increasing order.

```
{"command":"configure_function","callback":"ratemeter","params":{"section":0,"lemo_enables":[{"lemo":0,"enable":true},{"lemo":1,"enable":true},{"lemo":2,"enable":true},{"lemo":3,"enable":true}],"gate":false}}
```

PARAMETERS	VALID VALUES	FUNCTION
section	0/1/2/3	Section A/B/C/D of the instrument
lemo	0/1/2/3	Input channel number
enable	true/false	Enable/disable the correspondent input channel
gate	true/false	Enable/disable external gate

Table 3.15: table of the RATEMETER function configurable parameters.

Configure RATEMETER ADVANCED

This command allows to configure the RATEMETER ADVANCED function. The “lemo_enables” and the “thresholds” are arrays of four elements, corresponding to the input channels of the section, whose number should be in increasing order.

```
{"command":"configure_function","callback":"ratemeter_adv","params":{"section":1,"lemo_enables":[{"lemo":0,"enable":true},{"lemo":1,"enable":true},{"lemo":2,"enable":true},{"lemo":3,"enable":true}],"thresholds":[{"lemo":0,"threshold":1000},{"lemo":1,"threshold":1000},{"lemo":2,"threshold":1000},{"lemo":3,"threshold":1000}],"gate":false,"alarm":true,"filter":0,"int_time":3}}
```

PARAMETERS	VALID VALUES	FUNCTION
section	0/1/2/3	Section A/B/C/D of the instrument
lemo	0/1/2/3	Input channel number
enable	true/false	Enable/disable the correspondent input channel
gate	true/false	Enable/disable external gate
threshold	0…100000000	Alarm rate threshold in Hz
alarm	true/false	Enable/disable alarm on measured rate
filter	0/1/2/3/4/5	Measured rate filter mode: OFF/VERY SLOW/SLOW/MEDIUM/FAST/VERY FAST
int_time	0/1/2/3/4/5/6/7/8/9	Integration time for rate measurement: 1 ms/100 ms/500 ms/1 s/5 s/10 s/30 s/1 min/10 min/1 h

Table 3.16: table of the RATEMETER ADVANCED function configurable parameters.

Configure TIME TAGGING

This command allows to configure the TIME TAGGING function. The “lemo_enables” is an array of six elements, corresponding to the input channels of the section, whose number should be in increasing order.

```
{"command":"configure_function","callback":"time_tag","params":{"section":0,"lemo_enables":[{"lemo":0,"enable":true},{"lemo":1,"enable":true},{"lemo":2,"enable":true},{"lemo":3,"enable":true},{"lemo":4,"enable":true},{"lemo":5,"enable":true}]}}
```

PARAMETERS	VALID VALUES	FUNCTION
section	0/1/2/3	Section A/B/C/D of the instrument
lemo	0/1/2/3/4/5	Input channel number
enable	true/false	Enable/disable the correspondent input channel

Table 3.17: table of the TIME TAGGING function configurable parameters.

Configure TIME OF FLIGHT

There are three different ways to configure the TIME OF FLIGHT function. In order to measure the signals time of flight it is possible to use fixed time values, custom time values that can be defined in a specific file that has to be created, or custom time values that are described in a file that is already stored in the device. The “lemo_enables” is an array of six elements, corresponding to the input channels, whose number should be in increasing order.

```
{"command":"configure_function","callback":"tof","params":{"section":0,"lemo_enables":[{"lemo":0,"enable":true},{"lemo":1,"enable":true},{"lemo":2,"enable":true},{"lemo":3,"enable":true},{"lemo":4,"enable":true},{"lemo":5,"enable":true}],"win_mode":0,"win_value":10,"win_number":100,"t0_mode":0,"t0_value":1,"t0_reset":false}}
```

```
{"command":"configure_function","callback":"tof","params":{"section":0,"lemo_enables":[{"lemo":0,"enable":true},{"lemo":1,"enable":true},{"lemo":2,"enable":true},{"lemo":3,"enable":true},{"lemo":4,"enable":true},{"lemo":5,"enable":true}],"win_mode":1,"file_mode":1,"file_name":"tof","win_values":[{"window":0,"value":100},{"window":1,"value":200},{"window":2,"value":150},{"window":3,"value":100},{"window":4,"value":346}],"win_number":5,"t0_mode":0,"t0_value":1,"t0_reset":false}}
```

```
{"command":"configure_function","callback":"tof","params":{"section":0,"lemo_enables":[{"lemo":0,"enable":true},{"lemo":1,"enable":true},{"lemo":2,"enable":true},{"lemo":3,"enable":true},{"lemo":4,"enable":true},{"lemo":5,"enable":true}],"win_mode":1,"file_mode":0,"file_name":"tof","t0_mode":0,"t0_value":1,"t0_reset":false}}
```

PARAMETERS	VALID VALUES	FUNCTION
section	0/1/2/3	Section A/B/C/D of the instrument
lemo	0/1/2/3/4/5	Input channel number
enable	true/false	Enable/disable the correspondent input channel

win_mode	0/1	Use fixed/custom time windows
file_mode	0/1	Use an existing/create a new time window file
file_name	At maximum 20 characters. Characters allowed: letters, numbers, _, + and -	Time window file name to be used (existing file o file to be created)
win_value	10…1000000000	Fixed time window values in ns
win_values	[{},…,{}]	Content of the time window file to be created: array of elements containing the window index and its time value expressed in ns (valid range: 0-1000000000)
win_number	0…2048	Number of fixed or custom time windows
t0_mode	0/1	Time measurement starting mode: EXTERNAL (input channel)/INTERNAL
t0_value	10…1000000000	Frequency in Hz of the signal for the internal starting time mode
t0_reset	true/false	Enable/disable the timing measurement reset at starting signal occurrence

Table 3.18: table of the TIME OF FLIGHT function configurable parameters.

Configure TIME OVER THRESHOLD

This command allows to configure the TIME OVER THRESHOLD function. Only fixed time window values can be used in order to measure the signals time over threshold. The “lemo_enables” is an array of six elements, corresponding to the input channels, whose number should be in increasing order.

```
{"command":"configure_function","callback":"tot","params":{"section":0,"lemo_enables":[{
"lemo":0,"enable":true},{"lemo":1,"enable":true},{"lemo":2,"enable":true},{"lemo":3,"ena
ble":true},{"lemo":4,"enable":true},{"lemo":5,"enable":true}],"win_mode":0,"win_value":1
0,"win_number":100}}
```

PARAMETERS	VALID VALUES	FUNCTION
section	0/1/2/3	Section A/B/C/D of the instrument
lemo	0/1/2/3/4/5	Input channel number
enable	true/false	Enable/disable the correspondent input channel
win_value	10…1000000000	Fixed time window values in ns
win_number	0…1024	Number of fixed time windows

Table 3.19: table of the TIME OVER THRESHOLD function configurable parameters.

Configure PULSE GENERATOR

This command allows to configure the PULSE GENERATOR function. The “lemo_enables” is an array of four elements, corresponding to the output channels of the section, whose number should be in increasing order.

```
{"command":"configure_function","callback":"pulse_gen","params":{"section":0,"frequency_
type":0,"width":100,"frequency":100,"lemo_enables":[{"lemo":0,"enable":true},{"lemo":1,"
enable":true},{"lemo":2,"enable":true},{"lemo":3,"enable":true}]}}
```

PARAMETERS	VALID VALUES	FUNCTION
section	0/1/2/3	Section A/B/C/D of the instrument
lemo	0/1/2/3	Output channel number
enable	true/false	Enable/disable the correspondent output channel
frequency_type	0/1	Deterministic/Poisson output signal frequency
width	10…100000	Output signal width
frequency	1…100000000	Output signal frequency

Table 3.20: table of the PULSE GENERATOR function configurable parameters.

Configure DIGITAL GENERATOR

This command allows to configure the DIGITAL GENERATOR function. The “lemo_enables” is an array of four elements, corresponding to the output channels of the section, whose number should be in increasing order.

```
{"command":"configure_function","callback":"dig_gen","params":{"section":0,"lemo_enables
":[{"lemo":0,"enable":true},{"lemo":1,"enable":true},{"lemo":2,"enable":true},{"lemo":3,
"enable":true}]}}
```

PARAMETERS	VALID VALUES	FUNCTION
section	0/1/2/3	Section A/B/C/D of the instrument

lemo	0/1/2/3	Output channel number
enable	true/false	Enable/disable the correspondent output channel

Table 3.21: table of the DIGITAL GENERATOR function configurable parameters.

Configure PATTERN GENERATOR

The PATTERN GENERATOR function requires a file to define the sequence of values assumed by each output channel. It is possible to configure the function by passing the content of the file to be stored and used or by specifying the file name if it is already stored into the device. The "lemo_enables" is an array of four elements, corresponding to the output channels, whose number should be in increasing order.

```
{"command":"configure_function","callback":"pat_gen","params":{"section":0,"lemo_enables
":[{"lemo":0,"enable":true},{"lemo":1,"enable":true},{"lemo":2,"enable":true},{"lemo":3,
"enable":true}],"frequency":100,"file_mode":1,"file_name":"pattern","pattern_values":[{"
pattern":0,"value":10},{"pattern":1,"value":15},{"pattern":2,"value":2},{"pattern":3,"va
lue":4}],"total_number":4}}
```

```
{"command":"configure_function","callback":"pat_gen","params":{"section":0,"lemo_enables
":[{"lemo":0,"enable":true},{"lemo":1,"enable":true},{"lemo":2,"enable":true},{"lemo":3,
"enable":true}],"frequency":100,"file_mode":0,"file_name":"pattern"}}
```

PARAMETERS	VALID VALUES	FUNCTION
section	0/1/2/3	Section A/B/C/D of the instrument
lemo	0/1/2/3	Output channel number
enable	true/false	Enable/disable the correspondent output channel
frequency	1…100000000	Output pattern change frequency in Hz
file_mode	0/1	Use an existing/create a new pattern file
file_name	At maximum 20 characters. Characters allowed: letters, numbers, _, + and -	Pattern file name to be used (existing file or file to be created)
pattern_values	[{},…,{}]	Content of the pattern file to be created: array of elements containing the pattern index and its value describing the logic state of the output channels with a binary value expressed as decimal
total_number	Integer	Number of sequences defined in pattern file

Table 3.22: table of the PATTERN GENERATOR function configurable parameters.

Get Function Configuration

This command allows to obtain the configuration parameters of the function currently used in the specified section.

```
{"command":"get_function_config","callback":"pulse_gen","params":{"section":0}}
```

PARAMETERS	VALID VALUES	FUNCTION
section	0/1/2/3	Section A/B/C/D of the instrument

Table 3.23: table of the Get Function configurable parameters.

The retrieved information is organized in the following way, with the "data" containing all the function configuration parameters. The meaning of each parameter is described in the correspondent configuration function command. Here is an example for the PULSE GENERATOR function.

```
{"Response":"","Result":true,"callback":"pulse_gen","command":"get_function_config","dat
a":{"lemo_enables":[{"lemo":0,"enable":true},{"lemo":1,"enable":true},{"lemo":2,"enable"
:true},{"lemo":3,"enable":true}],"frequency_type":0,"frequency":100,"width":450}}
```

Get Function File List

This command allows to obtain the list of the configuration file stored in the device, related to a specific function.

```
{"command":"get_config_file","callback":"lut","function":"lut"}
```

PARAMETERS	VALID VALUES	FUNCTION
function	"lut"/"pattern"/ "width"	Name of the function whose configuration file name list has to be retrieved: LOOKUP TABLE/PATTERN GENERATOR/TIME OF FLIGHT

Table 3.24: table of the Get Function File List configurable parameters.

The retrieved information is organized in the following way, with the “data” containing the list of file names divided by a semicolon.

```
{"Response":"","Result":true,"callback":"lut","command":"get_config_file","data":"lut.json;lut2.json;test_file.json;"}
```

Get Function File

This command allows to obtain the list of the the content of the specified configuration file stored in the device related to a specific function.

```
{"command":"download_config","callback":"lut","params":{"file_name":"lut","function":"lut"}}
```

PARAMETERS	VALID VALUES	FUNCTION
file_name	At maximum 20 characters. Characters allowed: letters, numbers, _, + and -	Name of an existing configuration file of the specified function stored in the device to retrieve
function	"lut"/"pattern"/ "width"	Name of the function whose configuration file has to be retrieved: LOOKUP TABLE/PATTERN GENERATOR/TIME OF FLIGHT

Table 3.25: table of the Get Function File configurable parameters.

The retrieved information is organized in the following way, with the “data” containing the content of the function configuration file. Here is an example for the LOOKUP TABLE function.

```
{"Response":"","Result":true,"callback":"lut","command":"download_config","data":{"section":1,"lemo_out_enables":[{"lemo":0,"enable":true},{"lemo":1,"enable":true},{"lemo":2,"enable":true},{"lemo":3,"enable":true}],"lemo_in_enables":[{"lemo":0,"enable":true},{"lemo":1,"enable":true},{"lemo":2,"enable":true},{"lemo":3,"enable":true},{"lemo":4,"enable":true},{"lemo":5,"enable":true}],"file_mode":1,"file_name":"lut","lut_values":[{"input":63,"output":0},{"input":0,"output":15},{"input":21,"output":10},{"input":42,"output":5}],"total_number":4}}
```

Delete Function File

This command allows to delete from the device the specified configuration file stored in the device related to a specific function.

```
{"command":"delete_config","callback":"lut","params":{"file_name":"lut","function":"lut"}}
```

PARAMETERS	VALID VALUES	FUNCTION
file_name	At maximum 20 characters. Characters allowed: letters, numbers, _, + and -	Name of an existing configuration file of the specified function stored in the device to delete
function	"lut"/"pattern"/ "width"	Name of the function whose configuration file has to be retrieved: LOOKUP TABLE/PATTERN GENERATOR/TIME OF FLIGHT

Table 3.26: table of the Delete Function File configurable parameters.

Data Acquisition

Some functions generate data that can be retrieved and reset. In addition, some other functions require a start and a stop to manage the data acquisition or the signal generation. In all these commands it is specified only the section number and not the function name.

Get Function Data

The following command allows to get the acquired data from the COINCIDENCE GATE, SCALER, COUNTER, COUNTER TIMER, CHRONOMETER, RATEMETER, RATEMETER ADVANCED, TIME OF FLIGHT, TIME OVER THRESHOLD functions.

```
{"command":"get_function_results","callback":"coincidence_gate","params":{"section":0}}
```

PARAMETERS	VALID VALUES	FUNCTION
section	0/1/2/3	Section A/B/C/D of the instrument

Table 3.27: table of the Get Function Data configurable parameters.

The "data" of the retrieved information contains all the function measurements, which are different for every function.

For the COINCIDENCE GATE, the "counters" is an array of six elements: the first represents the total number of coincidences counts and the other are the coincidences counts on the correspondent input channel.

```
{"Response":"","Result":true,"callback":"coincidence_gate","command":"get_function_resul
ts","data":{"counters":[{"value":416},{"value":416},{"value":0},{"value":0},{"value":0},
{"value":0}]}}
```

For the SCALER, the "counters" is an array of four elements, each one consisting of a "lemo" index identifying the output channel and a "value" reporting the number of generated pulses for the corresponding output channel.

```
{"Response":"","Result":true,"callback":"scaler","command":"get_function_results","data"
:{"counters":[{"lemo":0,"value":10},{"lemo":1,"value":0},{"lemo":2,"value":35},{"lemo":3
,"value":1250}]}}
```

For the COUNTER, the "counters" is an array of four elements, each one consisting of a "lemo" index identifying the input channel and a "value" reporting the number of incoming pulses for the corresponding input channel.

```
{"Response":"","Result":true,"callback":"counter","command":"get_function_results","data
":{"counters":[{"lemo":0,"value":0},{"lemo":1,"value":10785},{"lemo":2,"value":0},{"lemo
":3,"value":39}]}}
```

For the COUNTER TIMER, the "counters" is an array of two elements, each one consisting of a "lemo" index identifying the input channel, a "value" reporting the measured counts for the corresponding input channel, a "target_value" indicating the target set for that channel and a "target_type" value defining the counter mode.

```
{"Response":"","Result":true,"callback":"counter_timer","command":"get_function_results"
,"data":{"counters":[{"lemo":0,"value":862,"target_value":1000,"target_type":2},{"lemo":
1,"value":356,"target_value":1000,"target_type":2}]}}
```

For the CHRONOMETER, the "counters" is an array of two elements, each one consisting of a "lemo" index identifying the input channel and a "value" reporting the measured counts for the corresponding input channel, which must be multiplied by ten in order to obtain the correspondent time expressed in ns.

```
{"Response":"","Result":true,"callback":"chronometer","command":"get_function_results","
data":{"counters":[{"lemo":0,"value":120},{"lemo":1,"value":500}]}}
```

For the RATEMETER, the "counters" is an array of four elements, each one consisting of a "lemo" index identifying the input channel and a "value" reporting the rate in Hz of incoming pulses for the corresponding input channel.

```
{"Response":"","Result":true,"callback":"ratemeter","command":"get_function_results","da
ta":{"counters":[{"lemo":0,"value":50},{"lemo":1,"value":0},{"lemo":2,"value":200},{"lem
o":3,"value":0}]}}
```

For the RATEMETER ADVANCED, the "counters" is an array of four elements, each one consisting of a "lemo" index identifying the input channel, a "value" reporting the rate in Hz of incoming pulses for the corresponding input channel and the "alarm" indicating if the rate threshold has been exceeded for that input channel.

```
{"Response":"","Result":true,"callback":"ratemater_adv","command":"get_function_results"
,"data":{"counters":[{"lemo":0,"value":100.000000,"alarm":false},{"lemo":1,"value":0.000
000,"alarm":false},{"lemo":2,"value":1000.000000,"alarm":false},{"lemo":3,"value":50.000
000,"alarm":false}]}}
```

For the TIME OF FLIGHT, the "data" contains five list of data, each one of a length equal to the number of time window used for the time measurement. The first four represents the time of flight distributions for the correspondent input channel, i.e. the number of measured times of flight in each time window. The last, the "time" list, represents the duration in ns of each time window.

{"Response":"","Result":true,"callback":"tof","command":"get_function_results","data":{"spectrum1":[0,100,0,0,0,0,0,0,0,0],"spectrum2":[0,0,0,0,0,0,0,0,0,0],"spectrum3":[0,0,10,22,89,243,101,64,18,0],"spectrum4":[0,0,0,0,0,0,0,0,0,0],"time":[10,10,10,10,10,10,10,10,10,10]}}

For the TIME OVER THRESHOLD, the “data” contains five list of data, each one of a length equal to the number of time window used for the time measurement. The first four represents the time over threshold distributions for the correspondent input channel, i.e. the number of measured signal duration for each time window. The last, the “time” list, represents the duration in ns of each time window.

{"Response":"","Result":true,"callback":"tot","command":"get_function_results","data":{"spectrum1":[0,0,0,0,10000,0,0,0,0,0],"spectrum2":[0,0,256,0,0,0,964,0,0,0],"spectrum3":[0,0,0,0,0,0,0,0,0,0],"spectrum4":[0,0,0,0,0,0,0,0,0,0],"time":[10,10,10,10,10,10,10,10,10,10]}}

In order to acquire data with the TIME TAGGING function, it is necessary to start the data acquisition, as shown in the ‘START FUNCTION DATA ACQUISITON’ paragraph. Then, it will be the device itself to send data when a web socket client is connected and the data are not null. In the sent JSON message the “timetag_data” consists of a list of couples of values, one representing the input channel index and the other is the time of arrival in ns of the pulse on that input channel.

{"command":"send_data","timetag_data":[[2,530],…,[1, 1870]]}

Reset Function Data

The following command allows to reset the function acquired data (count, rate or spectrum). For the COINCIDENCE GATE, TIME OF FLIGHT and TIME OVER THRESHOLD functions the command resets all the channels measurements, while with the SCALER, COUNTER, COUNTER TIMER, CHRONOMETER and RATEMETER ADVANCED function it is possible to specify the single input channel measurement to be reset.

{"command":"reset_channel","callback":"reset","params":{"section"0,"channel":0}}

PARAMETERS	VALID VALUES	FUNCTION
section	0/1/2/3	Section A/B/C/D of the instrument
channel	0/1/2/3	For the COINCIDENCE GATE: 0 For the TIME OF FLIGHT and TIME OVER THRESHOLD: 2 For the SCALER, COUNTER, COUNTER TIMER, CHRONOMETER and RATEMETER ADVANCED: correspondent input channel

Table 3.28: table of the Reset Function Data configurable parameters.

Start Function Data Acquisition

The following command allows to start the function data acquisition or the signal output generation. The functions that require this command are the LOOKUP TABLE, PATTERN GENERATOR, TIME TAGGING, TIME OF FLIGHT and TIME OVER THRESHOLD.

{"command":"reset_channel","callback":"start","params":{"section":0,"channel":1}}

PARAMETERS	VALID VALUES	FUNCTION
section	0/1/2/3	Section A/B/C/D of the instrument

Table 3.29: table of the Start Function Data Acquisition configurable parameters.

The TIME TAGGING function requires in addition the following command.

{"command":"start_tt_data","callback":"start","params":{"section":0}}

Stop Function Data Acquisition

The following command allows to stop the function data acquisition or the signal output generation. The functions that require this command are the LOOKUP TABLE, PATTERN GENERATOR, TIME TAGGING, TIME OF FLIGHT and TIME OVER THRESHOLD.

{"command":"reset_channel","callback":"stop","params":{"section":0,"channel":0}}

PARAMETERS	VALID VALUES	FUNCTION
section	0/1/2/3	Section A/B/C/D of the instrument

Table 3.30: table of the Start Function Data Acquisition configurable parameters.

The TIME TAGGING function requires in addition the following command.

```
{"command":"stop_tt_data","callback":"stop","params":{"section":0}}
```

I/Os Configuration

This section describes the commands used to set and to get the configuration of the input and output channels.

Set Input Configuration

The following command allows to configure the common parameters of the six inputs of each section.

```
{"command":"configure_input","callback":"input","params":{"section":0,"standard":1,"thre
shold":0,"imp":true}}
```

PARAMETERS	VALID VALUES	FUNCTION
section	0/1/2/3	Section A/B/C/D of the instrument
standard	0/1/2	Input signal standard: NIM/TTL/Analog with Voltage Threshold
threshold	0…2000	Threshold set on the analog input mode expressed in mV
imp	true/false	Input impedance: 50 Ohm/High impedance

Table 3.31: table of the Set Input configurable parameters.

Get Input Configuration

The following command allows to get the common configuration parameters of the six inputs of each section.

```
{"command":"get_input_config","callback":"input","params":{"section":0}}
```

The retrieved information is organized in the following way, with the “data” containing the input configuration status.

```
{"Response":"","Result":true,"callback":"input","command":"get_input_config","data":{"st
andard":0,"threshold":0,"imp":true}}
```

Set Input Channel Configuration

The following command allows to configure the specific parameters of an input channel.

```
{"command":"configure_input_channel","callback":"in_ch","params":{"section":0,"channel":
0,  "status":true,"enable_gd":true,"gate":200,"delay":100,"invert":false}}
```

PARAMETERS	VALID VALUES	FUNCTION
section	0/1/2/3	Section A/B/C/D of the instrument
channel	0/1/2/3/4/5	Input channel of the instrument section
status	true/false	Enable/disable the input channel
enable_gd	true/false	Enable/disable the input channel gate and delay
gate	0…100000	Gate value expressed in ns
delay	0…100000	Delay value expressed in ns
invert	true/false	Enable/disable the input channel signal inversion

Table 3.32: table of the Set Input Channel configurable parameters.

Get Input Channel Configuration

The following command allows to get the configuration parameters for a specific input channel.

```
{"command":"get_input_channel_config","callback":"in_ch","params":{"section":0,"channel"
:0}}
```

The retrieved information is organized in the following way, with the “data” containing the input channel configuration status.

```
{"Response":"","Result":true,"callback":"in_ch","command":"get_input_channel_config","da
ta":{"status":true,"enable_gd":false,"gate":0,"delay":0,"invert":false}}
```

Set Output Configuration

The following command allows to configure the common parameters of the four outputs of each section.

```
"command":"configure_output","callback":"output","params":{"section":0,"standard":1,"imp
":true}}
```

PARAMETERS	VALID VALUES	FUNCTION
section	0/1/2/3	Section A/B/C/D of the instrument
standard	0/1	Output signal standard: NIM/TTL
imp	true/false	Output impedance: 50 Ohm/High impedance

Table 3.33: table of the Set Output configurable parameters.

Get Output Configuration

The following command allows to get the common configuration parameters of the four outputs of each section.

```
{"command":"get_output_config","callback":"output","params":{"section":0}}
```

The retrieved information is organized in the following way, with the “data” containing the output configuration status.

```
{"Response":"","Result":true,"callback":"output","command":"get_output_config","data":{"
standard":1,"imp":true}}
```

Set Output Channel Configuration

The following command allows to configure the specific parameters of an output channel.

```
{"command":"configure_output_channel","callback":"out_ch","params":{"section":0,"channel
":0,  "status":true,"enable_mono":true,"mono_value":1000,"invert":false}}
```

PARAMETERS	VALID VALUES	FUNCTION
section	0/1/2/3	Section A/B/C/D of the instrument
channel	0/1/2/3	Output channel of the instrument section
status	true/false	Enable/disable the output channel
enable_mono	true/false	Enable/disable the output channel monostable
mono_value	0…1000	Monostable value expressed in ns
invert	true/false	Enable/disable the output channel signal inversion

Table 3.34: table of the Set Output Channel configurable parameters.

Get Input Channel Configuration

The following command allows to get the configuration parameters for a specific output channel.

```
{"command":"get_output_channel_config","callback":"out_ch","params":{"section":0,"channe
l":0}}
```

The retrieved information is organized in the following way, with the “data” containing the output channel configuration status.

```
{"Response":"","Result":true,"callback":"out_ch","command":"get_output_channel_config","
data":{"status":true,"enable_mono":false,"mono_value":0,"invert":false}}
```

I/Os Monitor

This section shows how to set the logic analyser in order to acquire the logic status of all the input and output channel of each section. In particular, the commands allow to set and get the configuration of the logic analyser trigger, to start the acquisition and to retrieve the data.

Set Logic Analyzer Trigger

This command allows to configure the trigger for the logic analyser acquisition. The “auto“ parameter should be equal to 0 to automatically trigger when the trigger condition is satisfied (self trigger). If the “mode_in” or “mode_out” is set

equal to 2 the values assigned to the enable of the input or output channels respectively are not taken into account. A logic OR is then applied between all the inputs and all the outputs trigger signals.


```
{"command":"configure_la_trigger","callback":"logic_analyser","params":{"file_content":"
{"mode_in":1,"mode_out":1,"edge":0,"auto":0,"sec1_in1":false,"sec1_in2":false,"sec1_in3"
:false,"sec1_in4":false,"sec1_in5":false,"sec1_in6":false,"sec1_out1":false,"sec1_out2":
false,"sec1_out3":false,"sec1_out4":false,"sec2_in1":false,"sec2_in2":false,"sec2_in3":f
alse,"sec2_in4":false,"sec2_in5":false,"sec2_in6":false,"sec2_out1":false,"sec2_out2":fa
lse,"sec2_out3":false,"sec2_out4":false,"sec3_in1":false,"sec3_in2":false,"sec3_in3":fal
se,"sec3_in4":false,"sec3_in5":false,"sec3_in6":false,"sec3_out1":true,"sec3_out2":true,
"sec3_out3":true,"sec3_out4":true,"sec4_in1":true,"sec4_in2":true,"sec4_in3":true,"sec4_
in4":true,"sec4_in5":true,"sec4_in6":true,"sec4_out1":false,"sec4_out2":false,"sec4_out3
":false,"sec4_out4":false"}}
```


PARAMETERS	VALID VALUES	FUNCTION
mode_in	0/1/2	Trigger mode applied to input channels: AND/OR/OFF
mode_out	0/1/2	Trigger mode applied to output channels: AND/OR/OFF
edge	0/1	Signal edge mode detected by trigger: RISING/FALLING
secX_inX	true/false	Input channel signal enabled/disabled for trigger
secX_outX	true/false	Output channel signal enabled/disabled for trigger

Table 3.35: table of the Set Logic Analyzer Trigger configurable parameters.

Get Logic Analyzer Trigger

This command allows to get the trigger configuration for the logic analyser acquisition.


```
{"command":"get_la_trigger_config","callback":"logic_analyser"}
```


The retrieved information is organized in the following way, with the “data” containing the trigger configuration status.


```
{"Response":"","Result":true,"callback":"logic_analyser","command":"get_la_trigger_confi
g","data":{"mode_in":1,"mode_out":1,"edge":0,"auto":0,"sec1_in1":false,"sec1_in2":false,
"sec1_in3":false,"sec1_in4":false,"sec1_in5":false,"sec1_in6":false,"sec2_in1":false,"se
c2_in2":false,"sec2_in3":false,"sec2_in4":false,"sec2_in5":false,"sec2_in6":false,"sec3_
in1":false,"sec3_in2":false,"sec3_in3":false,"sec3_in4":false,"sec3_in5":false,"sec3_in6
":false,"sec4_in1":true,"sec4_in2":true,"sec4_in3":true,"sec4_in4":true,"sec4_in5":true,
"sec4_in6":true,"sec1_out1":false,"sec1_out2":false,"sec1_out3":false,"sec1_out4":false,
"sec2_out1":false,"sec2_out2":false,"sec2_out3":false,"sec2_out4":false,"sec3_out1":true
,"sec3_out2":true,"sec3_out3":true,"sec3_out4":true,"sec4_out1":false,"sec4_out2":false,
"sec4_out3":false,"sec4_out4":false}}
```


Enable Logic Analyzer

This command allows to enable the logic analyser to start a single acquisition. Each time the user wants to acquire data new data from the logic analyser this command has to be sent.


```
{"command":"la_arm","callback":"logic_analyser"}
```


Get Logic Analyzer Data

This command allows to acquire data from the logic analyser.


```
{"command":"la_getdata","callback":"logic_analyser"}
```


The retrieved “data” information is divided in “inputs” and “outputs”. The “inputs” is composed by 24 arrays of 2048 values, representing the logic states of the input channels. The “outputs” consists of 16 arrays of 2048 values, representing the logic states of the output channels. The values can be 0 or 1 to indicate a logic low or high signal. The sampling rate of the logic analyser is 100 MHz.


```
{"Response":"","Result":true,"callback":"logic_analyser","command":"la_getdata","data":{
"inputs":[[0,…,0],…,[0,…,0]],"outputs":[[0,…,0],…,[0,…,0]]}}
```

Board General Settings

This section contains all the commands related to ethernet configuration, file settings configuration, alarm and clock status.

Set Ethernet Configuration

This command allows to configure the Ethernet parameters.

```
{"command":"set_eth_config","callback":"ethernet_config","data":{"dhcp":1,"ip":"10.128.0
.77","nm":"255.255.0.0","gw":"10.128.0.1","dns":"8.8.8.8"}}
```

PARAMETERS	VALID VALUES	FUNCTION
dhcp	0/1	Static Internet Protocol/Dynamic Host Configuration Protocol
ip	0-255.0-255.0-255.0-255	Internet Protocol Address specified in case of Static Internet Protocol
nm	0-255.0-255.0-255.0-255	Netmask specified in case of Static Internet Protocol
gw	0-255.0-255.0-255.0-255	Gateway specified in case of Static Internet Protocol
dns	0-255.0-255.0-255.0-255	Domain Name System specified in case of Static Internet Protocol

Table 3.36: table of the Set Ethernet configurable parameters.

Get Ethernet Configuration

This command allows to get the Ethernet configuration parameters.

```
{"command":"get_eth_config","callback":"ethernet_config"}
```

The retrieved information is organized in the following way, with the “data” containing the Ethernet configuration status.

```
{"Response":"","Result":true,"callback":"ethernet_config","command":"get_eth_config","da
ta":{"dhcp":1,"ip":"10.128.0.77","nm":"255.255.0.0","gw":"10.128.0.1","dns":"8.8.8.8"}}
```

Save Configuration File

This command allows to save the current configuration parameters in a json configuration file stored in the device, including the inputs, outputs and function settings for each section.

```
{"command":"create_config","callback":"cfg_file","params":{"new_name":"Config_test.json"
}}
```

PARAMETERS	VALID VALUES	FUNCTION
new_name	At maximum 20 characters. Characters allowed: letters, numbers, _, + and -	Name of the file (with .json extension specified) to be created into the device containing the current device configuration parameters

Table 3.37: table of the Save Configuration File configurable parameters.

Get Configuration File list

This command allows to get the list of the names of the configuration file stored in the instrument. The “function” is not a user defined parameter, but should be kept equal to “config”.

```
{"command":"get_config_file","callback":"cfg_file","function":"config"}
```

The retrieved information is organized in the following way, with the “data” containing a list of the stored configuration file names (with explicit extensions) divided by a semicolon.

```
{"Response":"","Result":true,"callback":"cfg_file","command":"get_config_file","data":"C
onfig_test.json;Config_Demo.json;Config_rma_pulse.json;Config_wire.json;"}
```

Load Configuration File

This command allows to set all the device configuration parameters as defined in the specified file stored in the device.

```
{"command":"load_config","callback":"cfg_file","params":{"file_name":"Config_test.json"}
}
```

PARAMETERS	VALID VALUES	FUNCTION
file_name	At maximum 20 characters. Characters allowed: letters, numbers, _, + and -	Name of the file (without .json extension specified) stored into the device containing the configuration parameters to be loaded

Table 3.38: table of the Load Configuration File configurable parameters.

Upload Configuration File

This command allows to load a configuration file containing all the required parameters to be stored in the device. The "function" is not a user defined parameter but should be kept equal to "config".

```
{"command":"upload_config","callback":"cfg_file","params":{"file_name":"Config_test.json
", "file_content":{…},"function":"config"}}
```

PARAMETERS	VALID VALUES	FUNCTION
file_name	At maximum 20 characters. Characters allowed: letters, numbers, _, + and -	Name of the file (without .json extension specified) to be created into the device containing the configuration parameters specified by the user in the "file_content"

Table 3.39: table of the Upload Configuration File configurable parameters.

The "file_content" is a .json file like this:

```
{"Section_0": {
"input_general": {
      "standard": 1,
      "threshold": 0,
      "imp": true
    },
    "input_channel_0": {
      "status": true,
      "enable_gd": false,
      "gate": 0,
      "delay": 0,
      "invert": false
    },…,
    "output_general": {
      "standard": 1,
      "imp": true
    },
    "output_channel_0": {
      "status": true,
      "enable_mono": false,
      "mono_value": 0,
      "invert": false
    },…,
    "function_name": "counter",
    "function_configuration": {
      "lemo_enables": [
        {
          "lemo": 0,
          "enable": true
        },
        {
          "lemo": 1,
```

```
          "enable": true
        },
        {
          "lemo": 2,
          "enable": true
        },
        {
          "lemo": 3,
          "enable": true
        }
      ],
      "gate": false
    }
  },
  "Section_1": {…},
  "Section_2": {…},
  "Section_3": {…}}
```


Download Configuration File

This command allows to get the content of a configuration file stored in the device. The “function” is not a user defined parameter, but should be kept equal to “config”.


```
{"command":"download_config","callback":"cfg_file","params":{"file_name":"Config_test.json", "function":"config"}}
```


PARAMETERS	VALID VALUES	FUNCTION
file_name	At maximum 20 characters. Characters allowed: letters, numbers, _, + and -	Name of the configuration file (without .json extension specified) stored into the device to be downloaded

Table 3.40: table of the Download Configuration File configurable parameters.

The retrieved information is organized in the following way, with the “data” containing all the configuration parameters of the specified file in a .json format:


```
{"Response":"","Result":true,"callback":"cfg_file","command":"download_config","data":{…}}
```


Delete Configuration File

This command allows to delete the specified configuration file stored in the device. The “function” is not a user defined parameter, but should be kept equal to “config”.


```
{"command":"delete_config","callback":"cfg_file","params":{"file_name":"Config_test.json", "function":"config"}}
```


PARAMETERS	VALID VALUES	FUNCTION
file_name	At maximum 20 characters. Characters allowed: letters, numbers, _, + and -	Name of the configuration file (without .json extension specified) stored into the device to be deleted

Table 3.41: table of the Delete Configuration File configurable parameters.

Rename Configuration File

This command allows to rename the specified configuration file stored in the device. The “function” is not a user defined parameter but should be kept equal to “config”.


```
{"command":"rename_config","callback":"cfg_file","params":{"old_name":"Config_test.json","new_name":"Config_test_new.json","function":"config"}}
```

PARAMETERS	VALID VALUES	FUNCTION
old_name	At maximum 20 characters. Characters allowed: letters, numbers, _, + and -	Current name of the configuration file (without .json extension specified) to be renamed into the device
new_name	At maximum 20 characters. Characters allowed: letters, numbers, _, + and -	New name of the configuration file (without .json extension specified) to be renamed into the device

Table 3.42: table of the Rename Configuration File configurable parameters.

Set Internal Clock

This command allows to set the internal clock as the device clock.

```
{"command":"apply_int_clk","callback":"clock"}
```

Set External Clock

This command allows to set the valid external clock signal as the device clock.

```
{"command":"apply_ext_clk","callback":"clock"}
```

Check External Clock

This command allows to check if the external provided signal is a valid clock signal.

```
{"command":"check_clk","callback":"clock"}
```

If the retrieved “data” value is “0” the external signal is not a valid clock. If the retrieved “data” value is “1” the external signal can be used as the device clock.

```
{"Response":"","Result":true,"callback":"clock","command":"check_clk","data":"0"}
```

Get Clock Status

This command allows to get the clock status.

```
{"command":"get_clk_status","callback":"clock"}
```

If the retrieved “data” value is “0” it means that the clock is set as external clock but the provided signal is no more valid and the system has switched to the internal clock. If the retrieved “data” value is “1” the system is using the external signal as a clock, while if the “data” value is “2” the system is using the internal clock.

```
{"Response":"","Result":true,"callback":"clock","command":"get_clk_status","data":"2"}
```

Get Device Version Info

This command allows to get the device serial number, the software, the zynq and the fpga versions.

```
{"command":"get_version","callback":"version"}
```

The retrieved information is organized in the following way, with the “data” containing all the versions information.

```
{"Response":"","Result":true,"callback":"version","command":"get_version","data":{"seria
l_number":"20","software_version":"2020.5.1.0","zynq_version":"19.10.15.01","fpga_versio
n":"18.10.09.00"}}
```

Start Search Device

This command allows to start the search device procedure, in which the device display becomes red and shows the serial number, while the device emits an alarm sound.

```
{"command":"start_alarm","callback":"search_device"}
```

Stop Search Device

This command allows to stop the search device procedure.

```
{"command":"stop_alarm","callback":"search_device"}
```

Get Search Device Status

This command allows to know if the device search procedure is active.

```
{"command":"get_alarm_status","callback":"search_device"}
```

If the retrieved “data” value is “0” the device is not in the search procedure, while if the “data” value is “1” the search device procedure is active.

```
{"Response":"","Result":true,"callback":"search_device","command":"get_alarm_status","da
ta":"0"}
```

4 Python SDK

A SDK for Python language is available. The SDK uses Web Socket API communication to interface with the N1081A.The SDK requires Python >3.2 and works both on x86/ia64/ARM processor. The SDK includes a library (N1081A_sdk.py) and an example file (N1081A_test_ sdk.py).

Required Modules

The SDK requires the following modules:

- `websocket [pip install websocket-client]`
- `enum [pip install enum]`

The example file requires the following modules:

- `pprint [pip install pprint]`
- `matplotlib [pip install matplotlib]`

Download the SDK

Python SDK files can be download from Nuclear Instruments Github:

https://github.com/NuclearInstruments/N1081A_SDK_Python

or cloned with git:

```
git clone https://github.com/NuclearInstruments/N1081A_SDK_Python.git
```

The SDK includes the library (N1081A_sdk.py) and an example file (N1081A_test_sdk.py)

Library Usage

In order to use the library, import the class N1081A and open a connection creating a new N1081A object:

```
from N1081A_sdk import N1081A
N1081A_device= N1081A("10.128.2.242")
```

In the class N1081A are defined other classes to help the user to insert the correct parameters to the library functions.

```
class FunctionType(Enum):
FN_WIRE = "wire"
FN_AND = "and"
FN_OR = "or"
FN_OR_VETO = "or_veto"
FN_VETO = "veto"
FN_MAJORITY = "majority"
FN_MAJORITY_VETO = "majority_veto"
FN_LUT = "lut"
FN_COINCIDENCE_GATE = "coincidence_gate"
FN_SCALER = "scaler"
FN_COUNTER = "counter"
FN_COUNTER_TIMER = "counter_timer"
FN_CHRONOMETER = "chronom"
FN_RATE_METER = "rate_meter"
FN_RATE_METER_ADVANCED = "rate_meter_advanced"
FN_TIME_TAG = "time_tag"
FN_TIME_OF_FLIGHT = "tof"
FN_TIME_OVER_THRESHOLD = "tot"
FN_PULSE_GENERATOR = "pulse_generator"
FN_DIGITAL_GENERATOR = "digital_generator"
FN_PATTERN_GENERATOR = "pattern_generator"
```

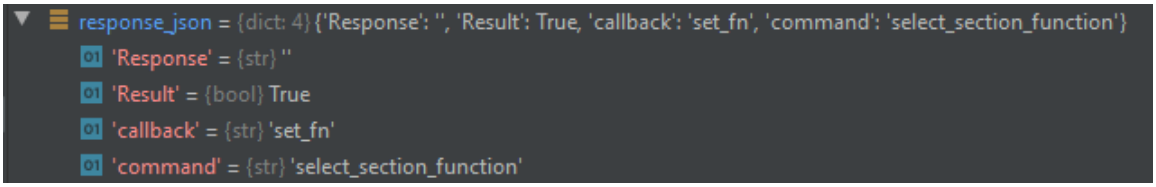
```
class Section(Enum):
SEC_A = 0
SEC_B = 1
SEC_C = 2
SEC_D = 3
```

In order to pass to a function a value defined in these classes, it is sufficient to refers to them in the following way:

```
N1081A.Section.SEC_A
```

Some other classes have been defined for other specific functions and will be shown later.

The response of all configuration functions that contain a set command is a dictionary of the following structure:



The following syntax allows to access a specific element of the dictionary (the “callback” string):

```
response_json["callback"]
```

The dictionary obtained as a response of a function that contains a get command has an additional component named “data” which is itself a dictionary. Its structure depends on the function and it is explained in the following.

Library Functions

The following functions are available:

- Set a function for a device section

```
set_section_function(self, section, function)
```

PARAMETERS	VALID VALUES	FUNCTION
section	N1081A.Section	Section A/B/C/D of the instrument
function	N1081A.FunctionType	Name of the function to be assigned to a section

Table 4.1: table of the parameters for `set_section_function`.

- Configure the WIRE function.

```
configure_wire(self, section, enable0, enable1, enable2, enable3)
```

PARAMETERS	VALID VALUES	FUNCTION
section	N1081A.Section	Section A/B/C/D of the instrument
enable0, enable1, enable2, enable3	bool	Enable/disable the correspondent input channel

Table 4.2: table of the parameters for `configure_wire`.

- Configure the AND function.

```
configure_and(self, section, enable0, enable1, enable2, enable3, enable4, enable5,
bypass_enable, bypass_ind)
```

PARAMETERS	VALID VALUES	FUNCTION
section	N1081A.Section	Section A/B/C/D of the instrument
enable0, enable1, enable2, enable3, enable4, enable5	bool	Enable/disable the correspondent input channel
bypass_enable	bool	Enable/disable the function bypass
bypass_ind	0/1/2/3/4	Bypass section OFF/A/B/C/D (No the current section)

Table 4.3: table of the parameters for `configure_and`.

- Configure the OR function.

```
configure_or(self, section, enable0, enable1, enable2, enable3, enable4, enable5,
bypass_enable, bypass_ind)
```

PARAMETERS	VALID VALUES	FUNCTION
section	N1081A.Section	Section A/B/C/D of the instrument
enable0, enable1, enable2, enable3, enable4, enable5	bool	Enable/disable the correspondent input channel
bypass_enable	bool	Enable/disable the function bypass
bypass_ind	0/1/2/3/4	Bypass section OFF/A/B/C/D (No the current section)

Table 4.4: table of the parameters for `configure_or`.

- Configure the OR+VETO function.

```
configure_or_veto(self, section, enable0, enable1, enable2, enable3, enable4,
bypass_enable, bypass_ind)
```

PARAMETERS	VALID VALUES	FUNCTION
section	N1081A.Section	Section A/B/C/D of the instrument
enable0, enable1, enable2, enable3, enable4	bool	Enable/disable the correspondent input channel
bypass_enable	bool	Enable/disable the function bypass
bypass_ind	int [0/1/2/3/4]	Bypass section OFF/A/B/C/D (No the current section)

Table 4.5: table of the parameters for `configure_or_veto`.

- Configure the VETO function.

```
configure_veto(self, section, enable0, enable1, enable2, enable3)
```

PARAMETERS	VALID VALUES	FUNCTION
section	N1081A.Section	Section A/B/C/D of the instrument
enable0, enable1, enable2, enable3	bool	Enable/disable the correspondent input channel

Table 4.6: table of the parameters for `configure_veto`.

- Configure the MAJORITY function.

```
configure_majority(self, section, enable0, enable1, enable2, enable3, enable4, enable5)
```

PARAMETERS	VALID VALUES	FUNCTION
section	N1081A.Section	Section A/B/C/D of the instrument
enable0, enable1, enable2, enable3, enable4, enable5	bool	Enable/disable the correspondent input channel

Table 4.7: table of the parameters for `configure_majority`.

- Configure the MAJORITY+VETO function.

```
configure_majority_veto(self, section, enable0, enable1, enable2, enable3, enable4)
```

PARAMETERS	VALID VALUES	FUNCTION
section	N1081A.Section	Section A/B/C/D of the instrument
enable0, enable1, enable2, enable3, enable4	bool	Enable/disable the correspondent input channel

Table 4.8: table of the parameters for `configure_majority_veto`.

- Configure the LUT function.

```
configure_lut(self, section, i_enable0, i_enable1, i_enable2, i_enable3, i_enable4,
i_enable5, o_enable0, o_enable1, o_enable2, o_enable3, file_mode, file_name,
file_content, file_lines)
```

PARAMETERS	VALID VALUES	FUNCTION
section	N1081A.Section	Section A/B/C/D of the instrument
i_enable0, i_enable1, i_enable2, i_enable3, i_enable4, i_enable5	bool	Enable/disable the correspondent input channel
o_enable0, o_enable1, o_enable2, o_enable3	bool	Enable/disable the correspondent output channel
file_mode	N1081A.FileMode	Use an existing/create a new LUT file
file_name	string [at maximum 20 characters. Characters allowed: letters, numbers, _, + and -]	LUT file name to be used (existing file or file to be created)
file_content	string '[{"input":X, "output":Y},…,{}]'	Content of the LUT file to be created: array of elements containing an input and an output value describing the logic state of the input and output channels with a binary value expressed as a decimal
file_lines	int	Number of combinations in the LUT file to be created

Table 4.9: table of the parameters for `configure_lut`.

```
class FileMode(Enum):

    FILE_EXISTING = 0

    FILE_NEW = 1
```

- Configure the COINCIDENCE GATE function.

```
configure_coincidence_gate(self, section, enable0, enable1, enable2, enable3, enable4,
coinc0, coinc1, coinc2, coinc3, coinc4, enable_gate, close_on_coinc, delay, width,
trigger_mode)
```

PARAMETERS	VALID VALUES	FUNCTION
section	N1081A.Section	Section A/B/C/D of the instrument
enable0, enable1, enable2, enable3, enable4	bool	Enable/disable the correspondent input channel
coinc0, coinc1, coinc2, coinc3, coinc4	bool	Enable coincidence/anticoincidence mode
enable_gate	bool	Enable/disable external gate
close_on_coinc	bool	Enable/disable closing of the coincidence gate when a coincidence occurs
delay	int [0…100000]	Delay of the coincidence gate in ns
width	int [0…100000]	Width of the coincidence gate in ns
trigger_mode	N1081A.CoincidenceTriggerMode	Coincidence gate opening mode: first arriving signal/signal of the correspondent input channel

Table 4.10: table of the parameters for `configure condicidence_gate`.

```
class CoincidenceTriggerMode(Enum):
        TRIGGER_FIRST = 0
        TRIGGER_1 = 1
        TRIGGER_2 = 2
        TRIGGER_3 = 3
        TRIGGER_4 = 4
        TRIGGER_5 = 5
```

- Configure the SCALER function.

```
configure_scaler(self, section, scale, enable0, enable1, enable2, enable3,
enable_gate)
```

PARAMETERS	VALID VALUES	FUNCTION
section	N1081A.Section	Section A/B/C/D of the instrument
scale	int [1…100000000]	Frequency divider value

enable0, enable1, enable2, enable3	bool	Enable/disable the correspondent input channel
enable_gate	bool	Enable/disable external gate

Table 4.11: table of the parameters for `configure scaler`.

- Configure the COUNTER function.

```
configure_counter(self, section, enable0, enable1, enable2, enable3, enable_gate)
```

PARAMETERS	VALID VALUES	FUNCTION
section	N1081A.Section	Section A/B/C/D of the instrument
enable0, enable1, enable2, enable3	bool	Enable/disable the correspondent input channel
enable_gate	bool	Enable/disable external gate

Table 4.12: table of the parameters for `configure counter`.

- Configure the COUNTER TIMER function.

```
configure_counter_timer(self, section, enable0, enable1, enable_gate, auto_reset,
gate1_width, gate2_width, source_mode, time, counter_mode, target1, target2)
```

PARAMETERS	VALID VALUES	FUNCTION
section	N1081A.Section	Section A/B/C/D of the instrument
enable0, enable1	bool	Enable/disable the correspondent input channel
enable_gate	bool	Enable/disable external gate
auto_reset	bool	Enable/disable the autoreset of the counts when the target is reached or the window is closed
gate1_width	int [0…2^32]	32 least significant bits of the window value expressed in decimal notation
gate2_width	int [0…2^32]	32 most significant bits of the window value expressed in decimal notation
source_mode	N1081A.CounterTimerSource	Input channel/Internal timing as counter source
time	N1081A.CounterTimerTime	Time value in case of internal timing counter source: 10 ns/1 us/1 ms/1 s
counter_mode	N1081A.CounterTimerMode	Counting mode: FREE/COUNTDOWN/TARGET/WINDOW
target1	int [0…2^32]	32 least significant bits of the target value expressed in decimal notation
target2	int [0…2^32]	32 most significant bits of the target value expressed in decimal notation

Table 4.13: table of the parameters for `configure counter timer`.

```
class CounterTimerSource(Enum):
        CT_INPUT = 0
        CT_TIME = 1

class CounterTimerTime(Enum):
        CT_10ns = 0
        CT_1us = 1
        CT_1ms = 2
        CT_1s = 3

class CounterTimerMode(Enum):
        CT_FREE = 0
        CT_COUNT_DOWN = 1
        CT_TARGET = 2
        CT_WINDOW = 3
```

- Configure the CHRONOMETER function.

```
configure_chronometer(self, section, frequency, chronometer_mode, enable0, enable1,
enable_gate, reset_on_gate, reset_on_stop)
```

PARAMETERS	VALID VALUES	FUNCTION
section	N1081A.Section	Section A/B/C/D of the instrument
frequency	int [1…100000000]	Output signal frequency in Hz
chronometer_mode	N1081A.ChronometerMode	Chronometer mode: GATE/START-STOP
enable0, enable1	bool	Enable/disable the correspondent input channel
enable_gate	Bool	Enable/disable external gate

reset_on_gate	bool	Enable/disable chronometer reset on gate closing in GATE mode
reset_on_stop	bool	Enable/disable chronometer reset on stop signal in START-STOP mode

Table 4.14: table of the parameters for `configure chronometer`.

```
class ChronometerMode(Enum):
        CM_GATE = 0
        CM_START_STOP = 1
```

- Configure the RATEMETER function.

```
configure_rate_meter(self, section, enable0, enable1, enable2, enable3, enable_gate)
```

PARAMETERS	VALID VALUES	FUNCTION
Section	N1081A.Section	Section A/B/C/D of the instrument
enable0, enable1, enable2, enable3	bool	Enable/disable the correspondent input channel
enable_gate	bool	Enable/disable external gate

Table 4.15: table of the parameters for `configure ratemeter`.

- Configure the RATEMETER ADVANCED function.

```
configure_rate_meter_advanced(self, section, enable0, enable1, enable2, enable3,
enable_gate, enable_alarm, threshold0, threshold1, threshold2, threshold3, filter_mode,
integration_time)
```

PARAMETERS	VALID VALUES	FUNCTION
section	N1081A.Section	Section A/B/C/D of the instrument
enable0, enable1, enable2, enable3	bool	Enable/disable the correspondent input channel
enable_gate	bool	Enable/disable external gate
enable_alarm	bool	Enable/disable alarm on measured rate
threshold0, threshold1, threshold2, threshold3	int [0…100000000]	Alarm rate threshold in Hz for each input channel
filter_mode	N1081A.FilterMode	Measured rate filter mode: OFF/VERY SLOW/SLOW/MEDIUM/FAST/VERY FAST
integration_time	N1081A.IntegrationTimeMode	Integration time for rate measurement: 1 ms/100 ms/500 ms/1 s/5 s/10 s/30 s/1 min/10 min/1 h

Table 4.16: table of the parameters for `configure ratemeter advanced`.

```
class FilterMode(Enum):
        FILTER_OFF = 0
        FILTER_VERY_SLOW = 1
        FILTER_SLOW = 2
        FILTER_MEDIUM = 3
        FILTER_FAST = 4
        FILTER_VERY_FAST = 5

class IntegrationTimeMode(Enum):
        TIME_1ms = 0
        TIME_100ms = 1
        TIME_500ms = 2
        TIME_1s = 3
        TIME_5s = 4
        TIME_10s = 5
        TIME_30s = 6
        TIME_1min = 7
        TIME_10min = 8
        TIME_1h = 9
```

- Configure the TIME TAGGING function.

```
configure_time_tagging(self, section, enable0, enable1, enable2, enable3, enable4,
enable5)
```

PARAMETERS	VALID VALUES	FUNCTION
section	N1081A.Section	Section A/B/C/D of the instrument
enable0, enable1, enable2, enable3, enable4, enable5	bool	Enable/disable the correspondent input channel

Table 4.17: table of the parameters for `configure time tagging`.

- Configure the TIME OF FLIGHT function.

```
configure_time_of_flight(self, section, enable0, enable1, enable3, enable4,
windows_mode, windows_value, windows_number, file_mode, file_name, file_content,
file_windows_number, t0_mode, t0_frequency, t0_reset)
```

PARAMETERS	VALID VALUES	FUNCTION
section	N1081A.Section	Section A/B/C/D of the instrument
enable0, enable1, enable3, enable4	bool	Enable/disable the correspondent input channel
windows_mode	N1081A.WindowsMode	Use fixed/custom time windows
windows_value	int [10…1000000000]	Fixed time window values in ns
windows_number	int [0…2048]	Number of fixed time windows
file_mode	N1081A.FileMode	Use an existing/create a new time window file
file_name	string [at maximum 20 characters. Characters allowed: letters, numbers, _, + and -]	Time window file name to be used (existing file o file to be created)
file_content	string ‘[{“window”:X, “value”:X},…,{}]’	Content of the time window file to be created: array of elements containing the window index and its time value expressed in ns (valid range: 0-1000000000)
file_windows_number	int [0…2048]	Number of custom time windows
t0_mode	N1081A.T0Mode	Time measurement starting mode: EXTERNAL (input channel)/INTERNAL
t0_frequency	int [10…1000000000]	Frequency in Hz of the signal for the internal starting time mode
t0_reset	bool	Enable/disable the timing measurement reset at starting signal occurrence

Table 4.18: table of the parameters for `configure time of flight`.

```
class WindowsMode(Enum):
        WINDOWS_FIXED = 0
        WINDOWS_CUSTOM = 1

class FileMode(Enum):
        FILE_EXISTING = 0
        FILE_NEW = 1

class T0Mode(Enum):
        T0_EXTERNAL = 0
        T0_INTERNAL = 1
```

- Configure the TIME OVER THRESHOLD function.

```
configure_time_over_threshold(self, section, enable0, enable1, enable3, enable4,
windows_value, windows_number)
```

PARAMETERS	VALID VALUES	FUNCTION
section	N1081A.Section	Section A/B/C/D of the instrument
enable0, enable1, enable3, enable4	bool	Enable/disable the correspondent input channel
windows_value	int [10…1000000000]	Fixed time window values in ns
windows_number	int [0…2048]	Number of fixed time windows

Table 4.19: table of the parameters for `configure time over threshold`.

- Configure the PULSE GENERATOR function.

```
configure_pulse_generator(self, section, statistics_mode, width, frequency, enable0,
enable1, enable2, enable3)
```

PARAMETERS	VALID VALUES	FUNCTION
section	N1081A.Section	Section A/B/C/D of the instrument
statistics_mode	N1081A.StatisticMode	Deterministic/Poisson output signal frequency
width	int [10…100000]	Output signal width
frequency	int [1…100000000]	Output signal frequency
enable0, enable1, enable2, enable3	bool	Enable/disable the correspondent output channel

Table 4.20: table of the parameters for `configure pulse generator`

```
class StatisticMode(Enum):
        STAT_DETERMINISTIC = 0
        STAT_POISSON = 1
```

- Configure the DIGITAL GENERATOR function.

```
configure_digital_generator(self, section, enable0, enable1, enable2, enable3)
```

PARAMETERS	VALID VALUES	FUNCTION
section	N1081A.Section	Section A/B/C/D of the instrument
enable0, enable1, enable2, enable3	bool	Enable/disable the correspondent output channel

Table 4.21: table of the parameters for `configure digital generator`

- Configure the PATTERN GENERATOR function.

```
configure_pattern_generator(self, section, enable0, enable1, enable2, enable3,
frequency, file_mode, file_name, file_content, file_lines)
```

PARAMETERS	VALID VALUES	FUNCTION
section	N1081A.Section	Section A/B/C/D of the instrument
enable0, enable1, enable2, enable3	bool	Enable/disable the correspondent output channel
frequency	int [1…100000000]	Output pattern change frequency in Hz
file_mode	N1081A.FileMode	Use an existing/create a new pattern file
file_name	string [at maximum 20 characters. Characters allowed: letters, numbers, _, + and -]	Pattern file name to be used (existing file or file to be created)
file_content	string '[{"pattern":X, "value":X},…,{}]'	Content of the pattern file to be created: array of elements containing the pattern index and its value describing the logic state of the output channels with a binary value expressed as decimal
file_lines	int	Number of sequences defined in pattern file

Table 4.22: table of the parameters for `configure pattern generator`

```
class FileMode(Enum):
        FILE_EXISTING = 0
        FILE_NEW = 1
```

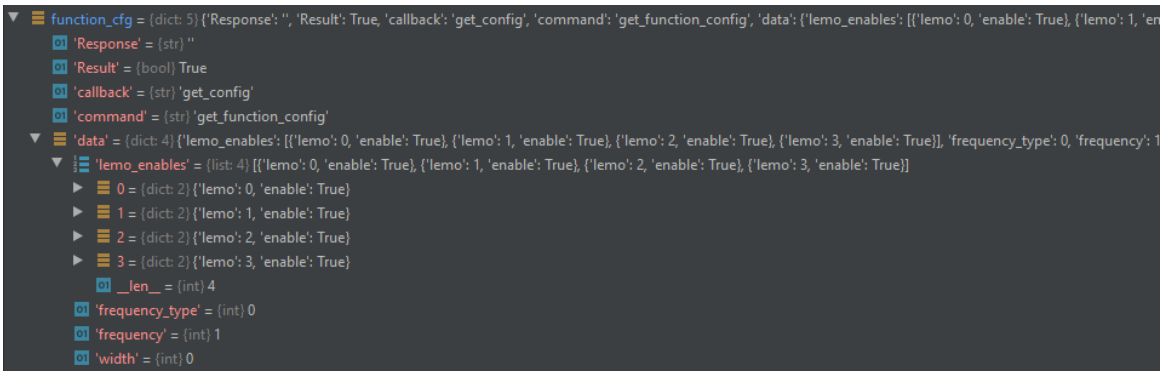
- Get the configuration parameters of the function currently used in the specified section.

```
get_function_configuration(self, section)
```

PARAMETERS	VALID VALUES	FUNCTION
section	N1081A.Section	Section A/B/C/D of the instrument

Table 4.23: table of the parameters for `get_function_configuration`

The function response is a dictionary containing all the function configuration parameters. The meaning of each parameter is described in the correspondent configuration function. Here is an example for the PULSE GENERATOR function.



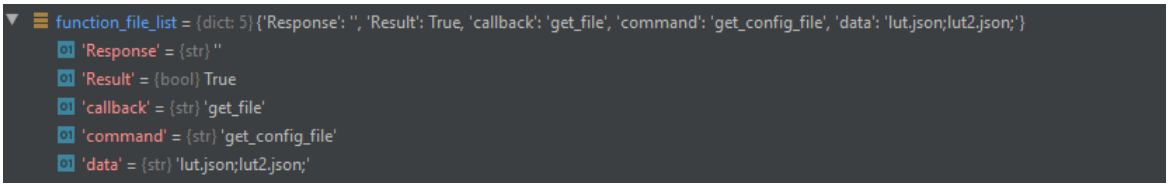
- Get the configuration parameters of the function currently used in the specified section.

```
get_function_file_list(self, function)
```

PARAMETERS	VALID VALUES	FUNCTION
function	N1081A.FunctionType.FN_LUT N1081A.FunctionType.FN_TIME_OF_FLIGHT N1081A.FunctionType.FN_PATTERN_GENERATOR	Name of the function whose configuration file name list has to be retrieved

Table 4.24: table of the parameters for `get_function_file_list`

The function response is a dictionary containing the list of the name of the files stored in the device of a specific function.



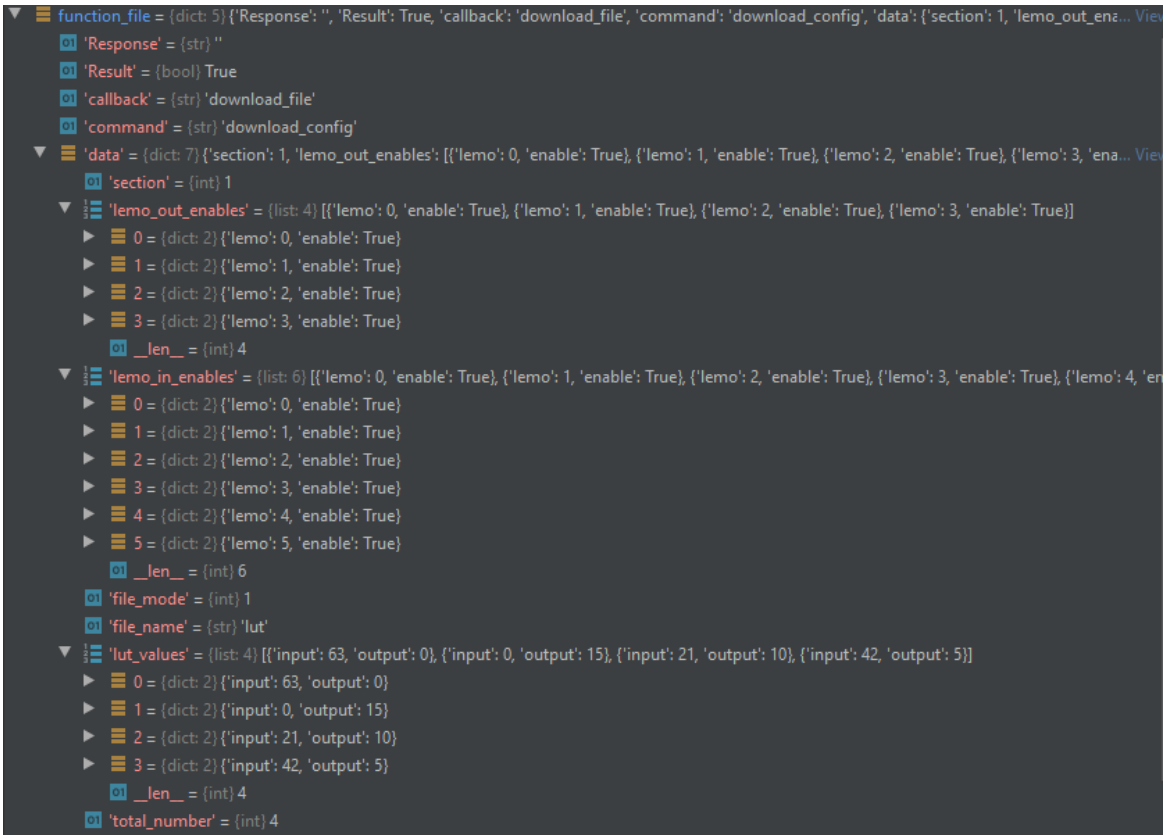
- Get the content of the specified configuration file stored in the device related to a specific function.

```
download_function_file(self, function, file_name)
```

PARAMETERS	VALID VALUES	FUNCTION
file_name	string [at maximum 20 characters. Characters allowed: letters, numbers, _, + and -]	Name of an existing configuration file of the specified function stored in the device to retrieve
function	N1081A.FunctionType.FN_LUT N1081A.FunctionType.FN_TIME_OF_FLIGHT N1081A.FunctionType.FN_PATTERN_GENERATOR	Name of the function whose configuration file has to be retrieved

Table 4.25: table of the parameters for `download_function_file`

The function response is a dictionary reporting the content of the specified configuration file of the desired function. Here is reported an example from the LUT function.



- Delete from the device the specified configuration file stored in the device related to a specific function.

```
delete_function_file(self, function, file_name)
```

PARAMETERS	VALID VALUES	FUNCTION
file_name	string [at maximum 20 characters. Characters allowed: letters, numbers, _, + and -]	Name of an existing configuration file of the specified function stored in the device to delete
function	N1081A.FunctionType.FN_LUT N1081A.FunctionType.FN_TIME_OF_FLIGHT N1081A.FunctionType.FN_PATTERN_GENERATOR	Name of the function whose configuration file has to be deleted

Table 4.26: table of the parameters for `delete_function_file`

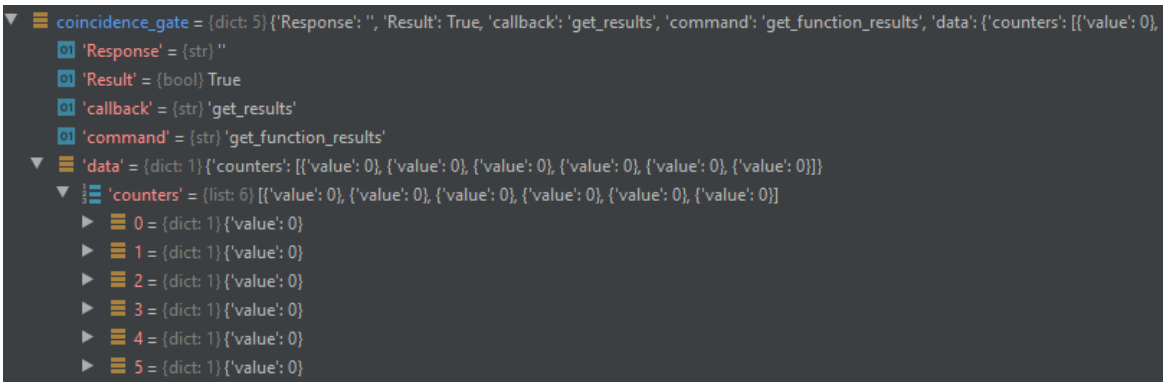
- Get the acquired data from the COINCIDENCE GATE, SCALER, COUNTER, COUNTER TIMER, CHRONOMETER, RATEMETER, RATEMETER ADVANCED, TIME OF FLIGHT, TIME OVER THRESHOLD functions.

```
get_function_results(self, section)
```

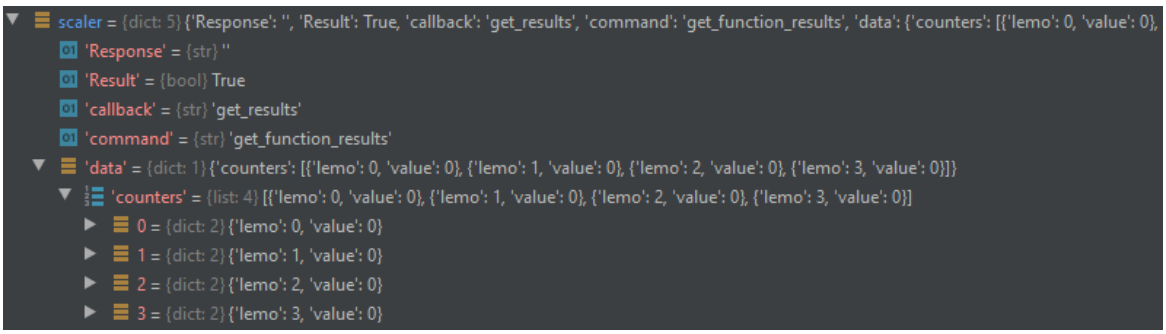
PARAMETERS	VALID VALUES	FUNCTION
section	N1081A.Section	Section A/B/C/D of the instrument

Table 4.27: table of the parameters for `get_function_results`

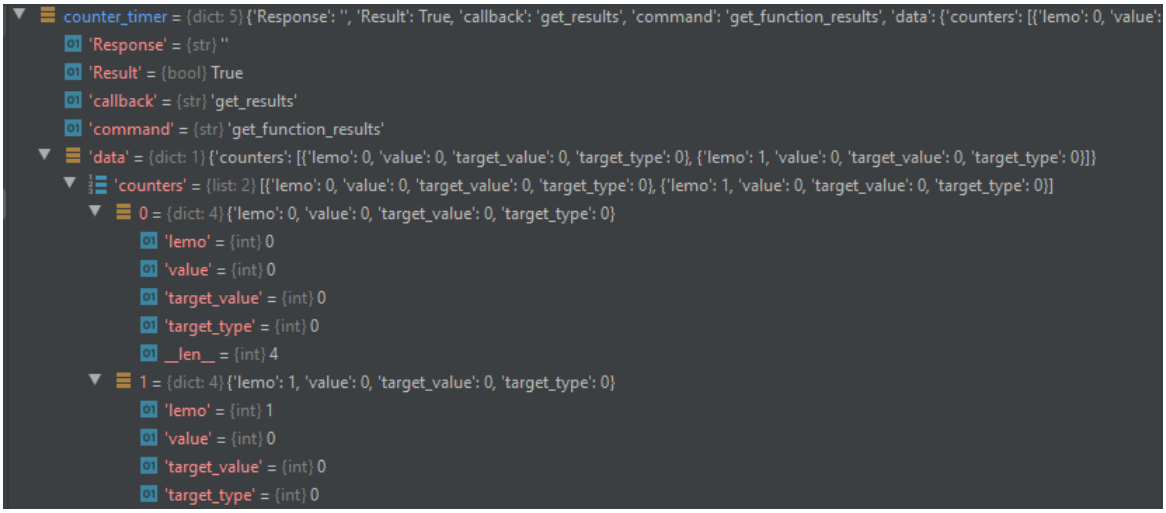
For the COINCIDENCE GATE, the response function is a dictionary containing six elements: the first represents the total number of coincidences counts and the other are the coincidences counts on the correspondent input channel.



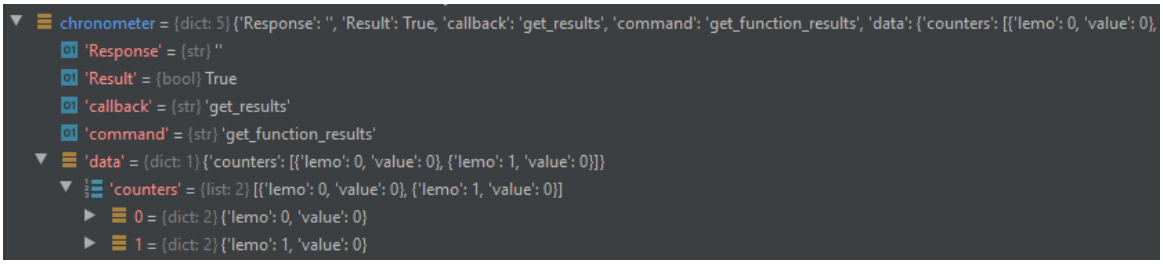
For the SCALER, COUNTER and RATEMETER, the response function is a dictionary containing four elements, each one consisting of a “lemo” index identifying the channel and a “value” reporting the counts/ rate for the corresponding channel.



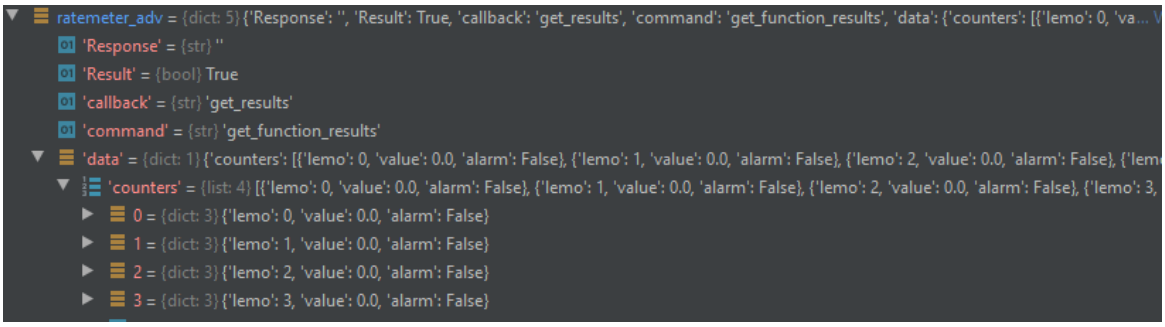
For the COUNTER TIMER, the response function is a dictionary containing two elements, each one consisting of a “lemo” index identifying the input channel, a “value” reporting the measured counts for the corresponding input channel, a “target_value” indicating the target set for that channel and a “target_type” value defining the counter mode.



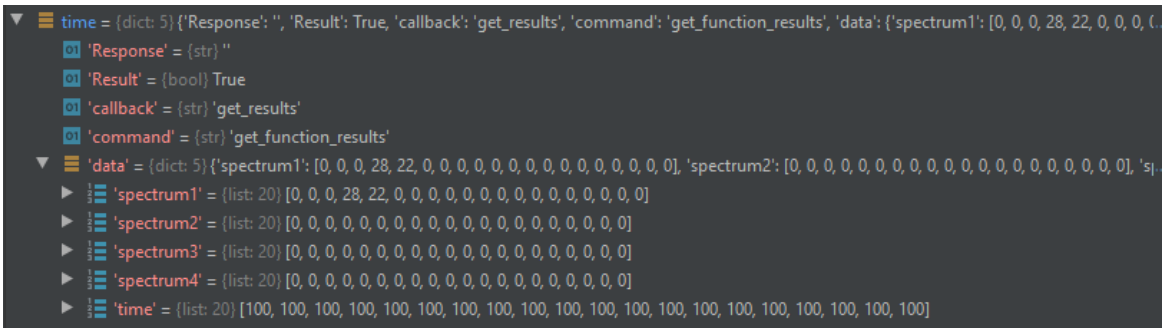
For the CHRONOMETER, the response function is a dictionary containing two elements, each one consisting of a “lemo” index identifying the input channel and a “value” reporting the measured counts for the corresponding input channel, which must be multiplied by ten in order to obtain the correspondent time expressed in ns.



For the RATEMETER ADVANCED, the response function is a dictionary containing four elements, each one consisting of a "lemo" index identifying the input channel, a "value" reporting the rate in Hz of incoming pulses for the corresponding input channel and the "alarm" indicating if the rate threshold has been exceeded for that input channel.



For the TIME OF FLIGHT and the TIME OVER THRESHOLD, the response function is a dictionary containing five list of data, each one of a length equal to the number of time window used for the time measurement. The first four represents the time of flight/ time over threshold distributions for the correspondent input channel, i.e. the number of measured times of flight/ signal duration in each time window. The last, the "time" list, represents the duration in ns of each time window.



- Reset the function acquired data. For the COINCIDENCE GATE, TIME OF FLIGHT and TIME OVER THRESHOLD functions all the channels measurements are reset, while for the SCALER, COUNTER, COUNTER TIMER, CHRONOMETER and RATEMETER ADVANCED functions it can be specified the input channel measurement to be reset

```
reset_channel(self, section, channel, function)
```

PARAMETERS	VALID VALUES	FUNCTION
section	N1081A.Section	Section A/B/C/D of the instrument
channel	0/1/2/3	Channel measurement to be reset for the SCALER, COUNTER, COUNTER TIMER, CHRONOMETER and RATEMETER ADVANCED
function	N1081A.FunctionType.FN_COINCIDENCE_GATE N1081A.FunctionType.FN_TIME_OF_FLIGHT N1081A.FunctionType.FN_TIME_OVER_THRESHOLD N1081A.FunctionType.FN_SCALER N1081A.FunctionType.FN_COUNTER N1081A.FunctionType.FN_COUNTER_TIMER N1081A.FunctionType.FN_CHRONOMETER N1081A.FunctionType.FN_RATE_METER_ADVANCED	Name of the function whose acquired data has to be reset

Table 4.28: table of the parameters for `reset_channel`.

- Start the function data acquisition or the signal output generation.

```
start_acquisition(self, section, function)
```

PARAMETERS	VALID VALUES	FUNCTION
section	N1081A.Section	Section A/B/C/D of the instrument
function	N1081A.FunctionType.FN_LUT N1081A.FunctionType.FN_TIME_TAGGING N1081A.FunctionType.FN_TIME_OF_FLIGHT N1081A.FunctionType.FN_TIME_OVER_THRESHOLD N1081A.FunctionType.FN_PATTERN_GENERATOR	Name of the function whose data acquisition or signal generation starts

Table 4.29: table of the parameters for `start_acquisition`.

- Stop the function data acquisition or the signal output generation.

```
stop_acquisition(self, section, function)
```

PARAMETERS	VALID VALUES	FUNCTION
section	N1081A.Section	Section A/B/C/D of the instrument
function	N1081A.FunctionType.FN_LUT N1081A.FunctionType.FN_TIME_TAGGING N1081A.FunctionType.FN_TIME_OF_FLIGHT N1081A.FunctionType.FN_TIME_OVER_THRESHOLD N1081A.FunctionType.FN_PATTERN_GENERATOR	Name of the function whose data acquisition or signal generation stops

Table 4.30: table of the parameters for `stop_acquisition`.

- Configure the common parameters of all the six inputs of each section.

```
set_input_configuration(self, section, standard, threshold, impedance)
```

PARAMETERS	VALID VALUES	FUNCTION
section	N1081A.Section	Section A/B/C/D of the instrument
standard	N1081A.SignalStandard	Input signal standard: NIM/TTL/Analog Voltage Threshold
threshold	int [0…2000]	Threshold set on the analog mode expressed in mV
impedance	bool	Input impedance: 50 Ohm/High impedance

Table 4.31: table of the parameters for `set_input_configuration`.

```
class SignalStandard(Enum):
        STANDARD_NIM = 0
        STANDARD_TTL = 1
        STANDARD_DISCRIMINATOR = 2
```

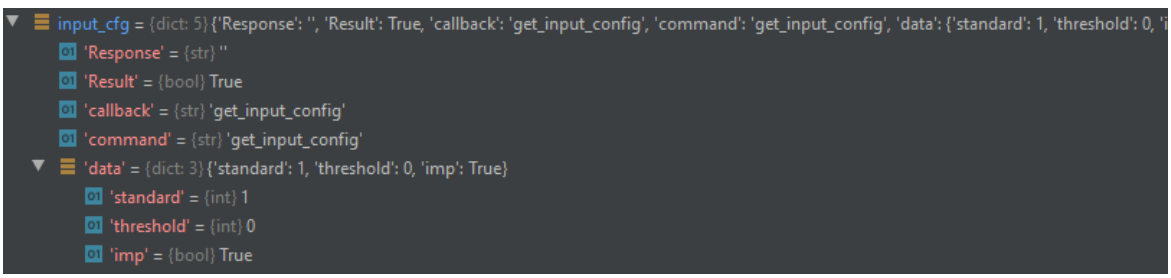
- Get the configuration parameters that are common for all the six inputs of each section.

```
get_input_configuration(self, section)
```

PARAMETERS	VALID VALUES	FUNCTION
section	N1081A.Section	Section A/B/C/D of the instrument

Table 4.32: table of the parameters for `get_input_configuration`.

The function response is a dictionary containing all the common input channel configuration parameters.



- Configure the specific parameters of an input channel.

```
set_input_channel_configuration(self, section, channel, status, enable_gate_delay, gate, delay, invert)
```

PARAMETERS	VALID VALUES	FUNCTION
section	N1081A.Section	Section A/B/C/D of the instrument
channel	0/1/2/3/4/5	Input channel of the instrument section
status	bool	Enable/disable the input channel
enable_gate_delay	bool	Enable/disable the input channel gate and delay
gate	int [0…100000]	Gate value expressed in ns
delay	int [0…100000]	Delay value expressed in ns
invert	bool	Enable/disable the input channel signal inversion

Table 4.33: table of the parameters for `set_input_channel_configuration`.

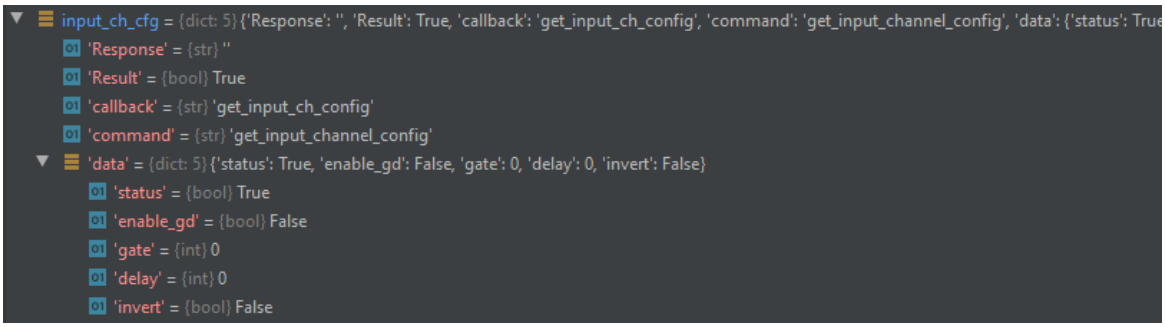
- Get the configuration parameters that are specific for an input channel.

```
get_input_channel_configuration(self, section, channel)
```

PARAMETERS	VALID VALUES	FUNCTION
section	N1081A.Section	Section A/B/C/D of the instrument
channel	0/1/2/3/4/5	Input channel of the instrument section

Table 4.34: table of the parameters for `get_input_channel_configuration`.

The function response is a dictionary containing all the specific input channel configuration parameters.



- Configure the common parameters of all the four outputs of each section.

```
set_output_configuration(self, section, standard, impedance)
```

PARAMETERS	VALID VALUES	FUNCTION
section	N1081A.Section	Section A/B/C/D of the instrument
standard	N1081A.SignalStandard	Output signal standard: NIM/TTL
impedance	bool	Output impedance: 50 Ohm/High impedance

Table 4.35: table of the parameters for `set_output_configuration`.

```
class SignalStandard(Enum):
        STANDARD_NIM = 0
        STANDARD_TTL = 1
        STANDARD_DISCRIMINATOR = 2
```

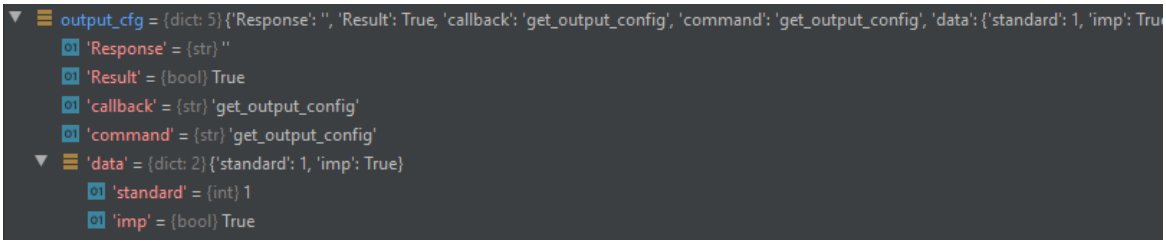
- Get the configuration parameters that are common for all the four outputs of each section.

```
get_output_configuration(self, section)
```

PARAMETERS	VALID VALUES	FUNCTION
section	N1081A.Section	Section A/B/C/D of the instrument

Table 4.36: table of the parameters for `get_output_configuration`.

The function response is a dictionary containing all the common output channel configuration parameters.



- Configure the specific parameters of an output channel.

```
set_output_channel_configuration(self, section, channel, status, enable_monostable, monostable, invert)
```

PARAMETERS	VALID VALUES	FUNCTION
section	N1081A.Section	Section A/B/C/D of the instrument
channel	0/1/2/3	Output channel of the instrument section
status	bool	Enable/disable the output channel
enable_monostable	bool	Enable/disable the output channel monostable
monostable	int [0…1000]	Monostable value expressed in ns
invert	bool	Enable/disable the output channel signal inversion

Table 4.37: table of the parameters for `set_output_channel_configuration`.

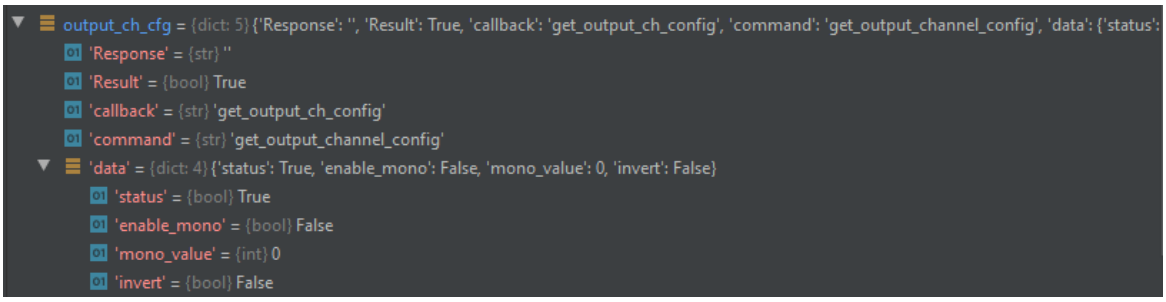
- Get the configuration parameters that are specific for an output channel.

```
get_output_channel_configuration(self, section, channel)
```

PARAMETERS	VALID VALUES	FUNCTION
section	N1081A.Section	Section A/B/C/D of the instrument
channel	0/1/2/3	Output channel of the instrument section

Table 4.38: table of the parameters for `get_output_channel_configuration`.

The function response is a dictionary containing all the specific output channel configuration parameters.



- Configure the trigger for the logic analyser acquisition. If the “trigger_mode_in” or “trigger_mode_out” is set to OFF the values assigned to the enable of the input or output channels respectively are not taken into account. A logic OR is then applied between all the inputs and all the outputs trigger signals.

```
set_logic_analyzer_trigger(self, trigger_mode_in, trigger_mode_out, edge, sec1_in1, sec1_in2, sec1_in3, sec1_in4, sec1_in5, sec1_in6, sec1_out1, sec1_out2, sec1_out3, sec1_out4, sec2_in1, sec2_in2, sec2_in3, sec2_in4, sec2_in5, sec2_in6, sec2_out1, sec2_out2, sec2_out3, sec2_out4, sec3_in1, sec3_in2, sec3_in3, sec3_in4, sec3_in5, sec3_in6, sec3_out1, sec3_out2, sec3_out3, sec3_out4, sec4_in1, sec4_in2, sec4_in3, sec4_in4, sec4_in5, sec4_in6, sec4_out1, sec4_out2, sec4_out3, sec4_out4)
```

PARAMETERS	VALID VALUES	FUNCTION
trigger_mode_in	N1081A.LogicAnalyzerTriggerMode	Trigger mode applied to input channels

trigger_mode_out	N1081A.LogicAnalyzerTriggerMode	Trigger mode applied to output channels
edge	N1081A.LogicAnalyzerTriggerEdge	Signal edge mode detected by trigger
secX_inX	bool	Input channel signal enabled/disabled for trigger
secX_outX	bool	Output channel signal enabled/disabled for trigger

Table 4.39: table of the parameters for `set_logic_analyzer_trigger`.

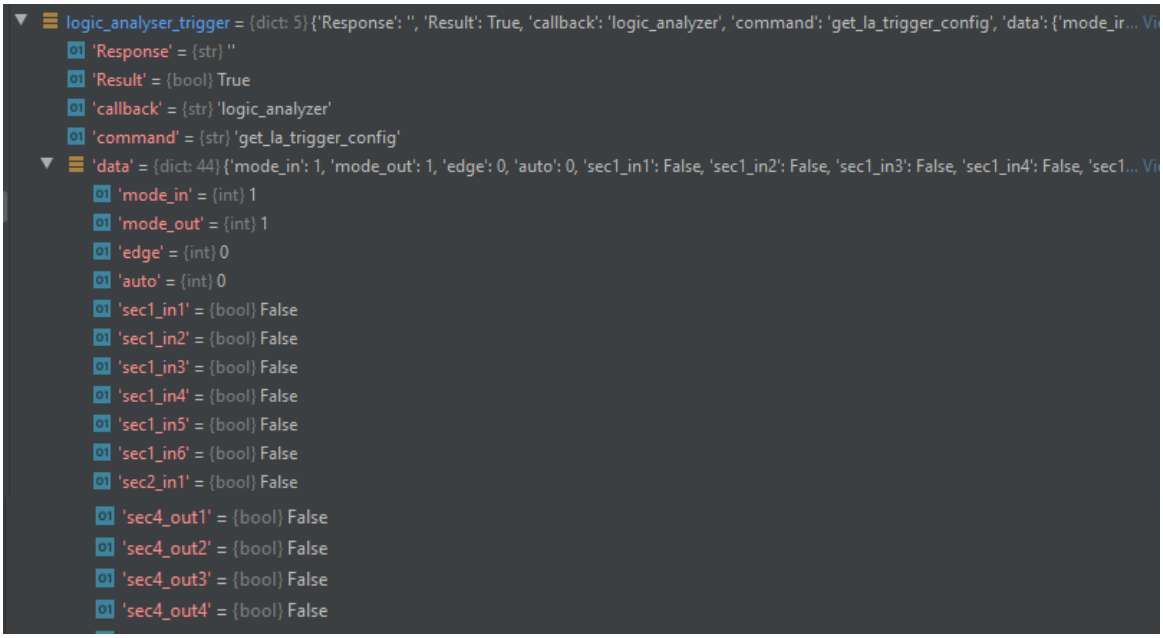
```
class LogicAnalyzerTriggerMode(Enum):
        LA_TRIGGER_AND = 0
        LA_TRIGGER_OR = 1
        LA_TRIGGER_OFF = 2

class LogicAnalyzerTriggerEdge(Enum):
        LA_EDGE_RISING = 0
        LA_EDGE_FALLING = 1
```

- Get the trigger configuration for the logic analyser acquisition.

```
get_logic_analyzer_trigger(self)
```

The function response is a dictionary containing all the logic analyser trigger parameters, whose meaning is explained in the 'set_logic_analyzer_trigger' function.



- Enable the logic analyser to start a single acquisition. Each time the user wants to acquire new data from the logic analyser this command has to be sent.

```
start_logic_analyzer(self)
```

- Acquire data from the logic analyser.

```
get_logic_analyzer_data(self)
```

The function response is a dictionary containing "inputs" and "outputs" lists. The "inputs" is composed by 24 lists of 2048 values, representing the logic states of the input channels. The "outputs" consists of 16 lists of 2048 values, representing the logic states of the output channels. The values can be 0 or 1 to indicate a logic low or high signal. The sampling rate of the logic analyser is 100 MHz.

```
▼ logic_analyser_data = {dict: 4} {'Result': True, 'callback': 'logic_analyzer', 'command': 'la_getdata', 'data': {'inputs': [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, (... View
    'Result' = {bool} True
    'callback' = {str} 'logic_analyzer'
    'command' = {str} 'la_getdata'
  ▼ 'data' = {dict: 2} {'inputs': [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ... View
    ▶ 'inputs' = {list: 28} [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,... View
    ▼ 'outputs' = {list: 16} [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ... View
      ▶ 00 = {list: 2048} [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,... View
      ▶ 01 = {list: 2048} [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,... View
      ▶ 02 = {list: 2048} [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,... View
      ▶ 03 = {list: 2048} [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,... View
      ▶ 04 = {list: 2048} [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,... View
      ▶ 05 = {list: 2048} [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,... View
      ▶ 06 = {list: 2048} [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,... View
      ▶ 07 = {list: 2048} [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,... View
      ▶ 08 = {list: 2048} [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,... View
      ▶ 09 = {list: 2048} [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,... View
      ▶ 10 = {list: 2048} [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,... View
      ▶ 11 = {list: 2048} [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,... View
      ▶ 12 = {list: 2048} [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,... View
      ▶ 13 = {list: 2048} [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,... View
      ▶ 14 = {list: 2048} [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,... View
      ▶ 15 = {list: 2048} [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,... View
```


- Configure the ethernet parameters.

```
set_ethernet_configuration(self, dhcp, ip, netmask, gateway, dns)
```

PARAMETERS	VALID VALUES	FUNCTION
`dhcp`	`0/1`	Static Internet Protocol/Dynamic Host Configuration Protocol
`ip`	`0-255.0-255.0-255.0-255`	Internet Protocol Address specified in case of Static Internet Protocol
`netmask`	`0-255.0-255.0-255.0-255`	Netmask specified in case of Static Internet Protocol
`gateway`	`0-255.0-255.0-255.0-255`	Gateway specified in case of Static Internet Protocol
`dns`	`0-255.0-255.0-255.0-255`	Domain Name System specified in case of Static Internet Protocol

Table 4.40: table of the parameters for `set_ethernet_configuration`

- Get the ethernet configuration parameters.

```
get_ethernet_configuration(self)
```

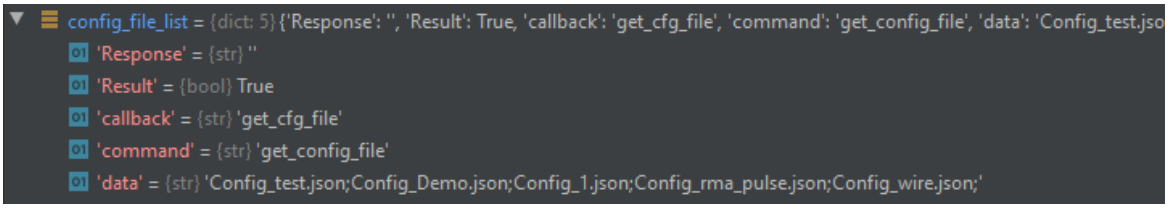
The function response is a dictionary containing the ethernet parameters.


```
▼ eth_config = {dict: 5} {'Response': '', 'Result': True, 'callback': 'ethernet_config', 'command': 'get_eth_config', 'data': {'dhcp': 1, 'ip':
    'Response' = {str} ''
    'Result' = {bool} True
    'callback' = {str} 'ethernet_config'
    'command' = {str} 'get_eth_config'
  ▼ 'data' = {dict: 5} {'dhcp': 1, 'ip': '10.128.0.77', 'nm': '255.255.0.0', 'gw': '10.128.0.1', 'dns': '8.8.8.8'}
      'dhcp' = {int} 1
      'ip' = {str} '10.128.0.77'
      'nm' = {str} '255.255.0.0'
      'gw' = {str} '10.128.0.1'
      'dns' = {str} '8.8.8.8'
```


- Get the list of the names of the configuration file stored in the instrument.

```
get_configuration_file_list(self)
```

The function response is a dictionary containing the list of the name of the configuration files stored in the device.



- Save the current configuration parameters in a json configuration file stored in the device, including the inputs, outputs and function settings for each section.

```
save_configuration_file(self, file_name)
```

PARAMETERS	VALID VALUES	FUNCTION
file_name	string [at maximum 20 characters. Characters allowed: letters, numbers, _, + and -]	Name of the file (with .json extension specified) to be created into the device containing the current device configuration parameters

Table 4.41: table of the parameters for `save_configuration_file`

- Set all the device configuration parameters as defined in the specified file stored in the device.

```
load_configuration_file(self, file_name)
```

PARAMETERS	VALID VALUES	FUNCTION
file_name	string [at maximum 20 characters. Characters allowed: letters, numbers, _, + and -]	Name of the file (without .json extension specified) stored into the device containing the configuration parameters to be loaded

Table 4.42: table of the parameters for `load_configuration_file`

- Load a configuration file containing all the required parameters to be stored in the device.

```
upload_configuration_file(self, file_name, file_content)
```

PARAMETERS	VALID VALUES	FUNCTION
file_name	string [at maximum 20 characters. Characters allowed: letters, numbers, _, + and -]	Name of the file (without .json extension specified) to be created into the device containing the configuration parameters specified by the user in the “file_content”

Table 4.43: table of the parameters for `upload_configuration_file`

The “file_content” is a string like this:

‘{"Section_0": { "input_general": { "standard": 1, "threshold": 0, "imp": true }, "input_channel_0": { "status": true, "enable_gd": false, "gate": 0, "delay": 0, "invert": false },…, "output_general": { "standard": 1, "imp": true }, "output_channel_0": { "status": true, "enable_mono": false, "mono_value": 0, "invert": false },…, "function_name": "counter", "function_configuration": { "lemo_enables": [{ "lemo": 0, "enable": true }, { "lemo": 1, "enable": true }, { "lemo": 2, "enable": true }, { "lemo": 3, "enable": true }], "gate": false } }, "Section_1": {…}, "Section_2": {…}, "Section_3": {…}}’

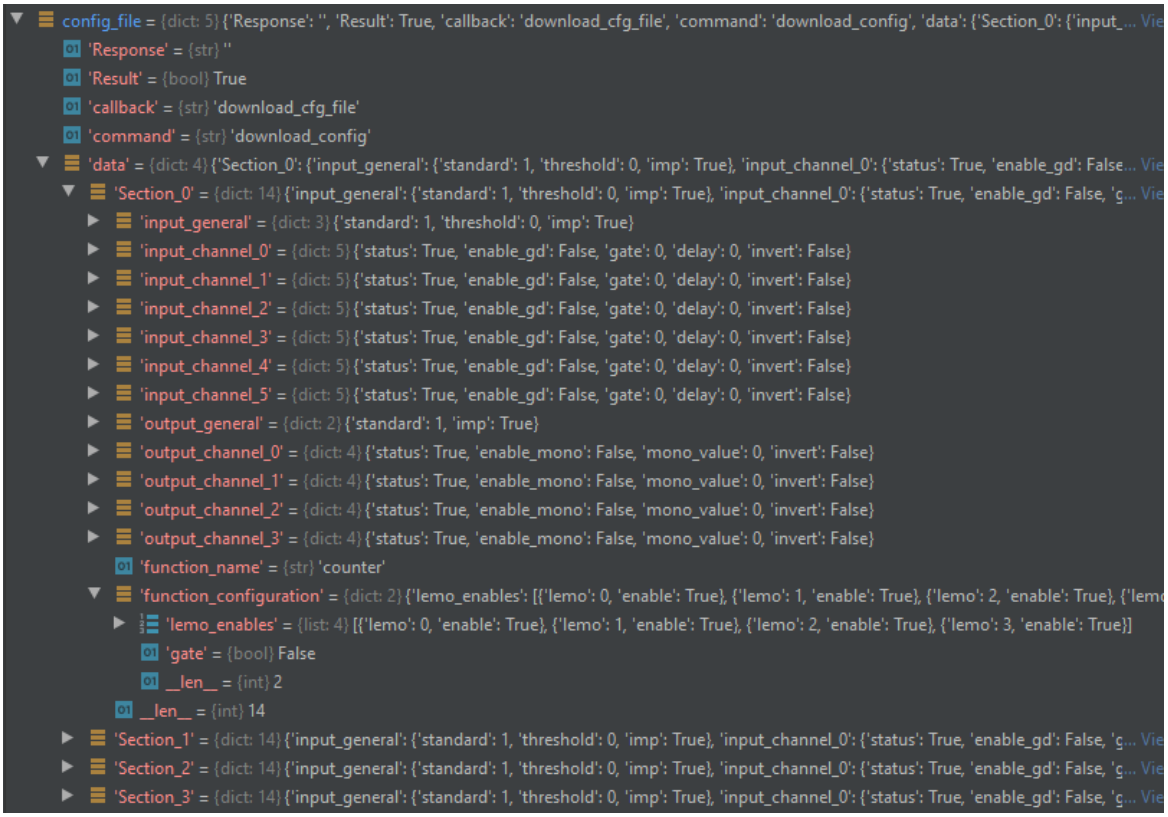
- Get the content of a configuration file stored in the device.

```
download_configuration_file(self, file_name)
```

PARAMETERS	VALID VALUES	FUNCTION
file_name	string [at maximum 20 characters. Characters allowed: letters, numbers, _, + and -]	Name of the configuration file (without .json extension specified) stored into the device to be downloaded

Table 4.44: table of the parameters for `download_configuration_file`

The function response is a dictionary reporting the content of the specified configuration file stored in the device.



- Delete the specified configuration file stored in the device.

```
delete_configuration_file(self, file_name)
```

PARAMETERS	VALID VALUES	FUNCTION
file_name	string [at maximum 20 characters. Characters allowed: letters, numbers, _, + and -]	Name of the configuration file (without .json extension specified) stored into the device to be deleted

Table 4.45: table of the parameters for delete_configuration_file

- Rename the specified configuration file stored in the device.

```
rename_configuration_file(self, old_file_name, new_file_name)
```

PARAMETERS	VALID VALUES	FUNCTION
old_file_name	string [at maximum 20 characters. Characters allowed: letters, numbers, _, + and -]	Current name of the configuration file (without .json extension specified) to be renamed into the device
new_file_name	string [at maximum 20 characters. Characters allowed: letters, numbers, _, + and -]	New name of the configuration file (without .json extension specified) to be renamed into the device

Table 4.46: table of the parameters for rename_configuration_file

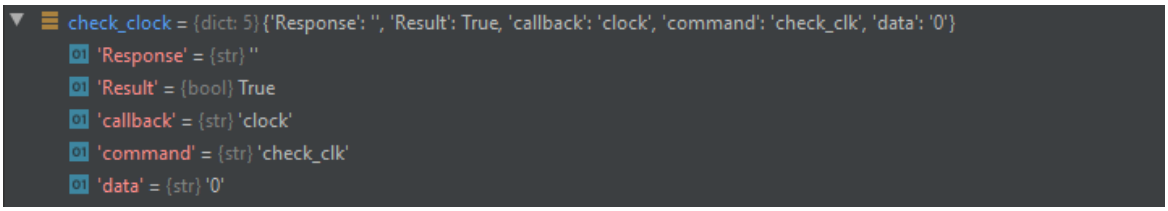
- Set the internal clock as the device clock.

```
set_internal_clock(self)
```

- Check if the external provided signal is a valid clock signal.

```
check_clock(self)
```

The function response is a dictionary where the “data” value represents the signal validity: if it is “0” the external signal is not a valid clock, if it is “1” the external signal can be used as the device clock.



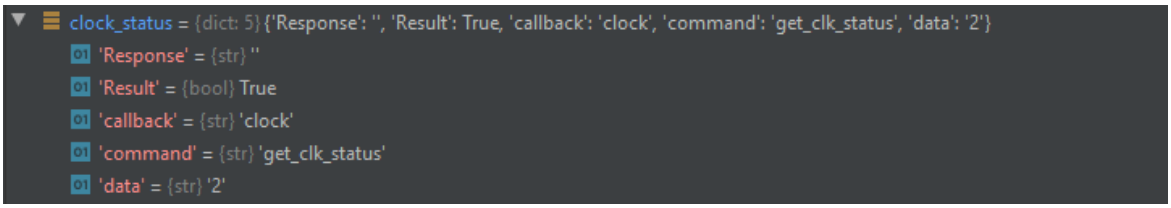
- Set the valid external clock signal as the device clock.

```
set_external_clock(self)
```

- Get the clock status.

```
get_clock_status(self)
```

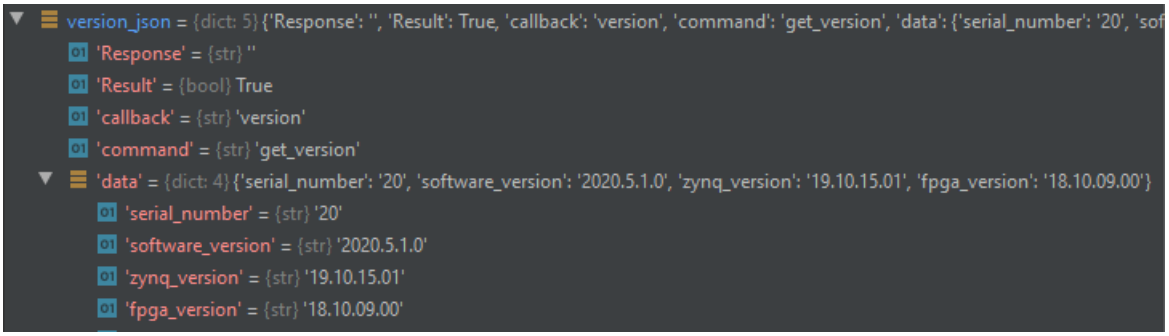
The function response is a dictionary where the “data” value represents the clock status: if it is “0” the clock is set as external clock but the provided signal is no more valid and the system has switched to the internal clock, if it is “1” the system is using the external signal as a clock, while if it is “2” the system is using the internal clock.



- Get the device serial number, the software, the zynq and the fpga versions.

```
get_version(self)
```

The function response is a dictionary containing the device serial number, the software, the zynq and the fpga versions.



- Start the search device procedure, in which the device display becomes red and shows the serial number, while the device emits an alarm sound.

```
start_search_device(self)
```

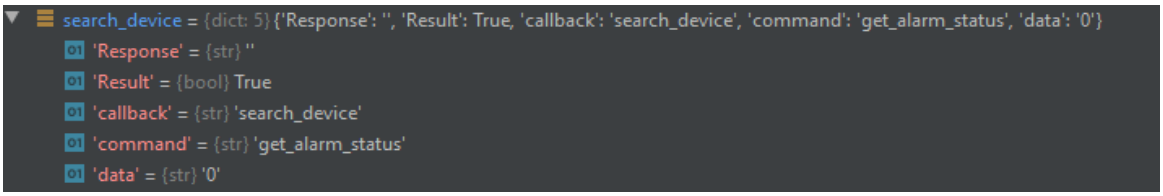
- Stop the search device procedure.

```
stop_search_device(self)
```

- Get the device search procedure status.

```
get_search_device_status(self)
```

The function response is a dictionary where the “data” value represents the search device status: if it is “0” the device is not in the search procedure, while if it is “1” the search device procedure is active.



5 Technical Support

CAEN makes available the technical support of its specialists for request concerning the software and the hardware.
Use the support form available at the following link:

https://www.caen.it/support-services/support-form/

CAEN SpA is acknowledged as the only company in the world providing a complete range of High/Low Voltage Power Supply systems and Front-End/Data Acquisition modules which meet IEEE Standards for Nuclear and Particle Physics. Extensive Research and Development capabilities have allowed CAEN SpA to play an important, long term role in this field. Our activities have always been at the forefront of technology, thanks to years of intensive collaborations with the most important Research Centres of the world. Our products appeal to a wide range of customers including engineers, scientists and technical professionals who all trust them to help achieve their goals faster and more effectively.

CAEN S.p.A.
Via Vetraia, 11
55049 Viareggio
Italy
Tel. +39.0584.388.398
Fax +39.0584.388.959
info@caen.it
www.caen.it

CAEN GmbH
Klingenstraße 108
D-42651 Solingen
Germany
Tel. +49 (0)212 254 4077
Mobile +49 (0)151 16 548 484
Fax +49 (0)212 25 44079
info@caen-de.com
www.caen-de.com
CAEN GmbH

CAEN Technologies, Inc.
1140 Bay Street - Suite 2 C
Staten Island, NY 10305
USA
Tel. +1.718.981.0401
Fax +1.718.556.9185
info@caentechnologies.com
www.caentechnologies.com

CAEN
Tools for Discovery
Electronic Instrumentation

UM7615 - N1081A Programmable Logic Unit User Manual Rev. 2 - December 23rd, 2020
00000-00-1081A-MUTX