A Passion For Precision

TeraScan / TeraBeam
Remote Command Reference

Firmware 3.1.1

TOPTICA Photonics AG

Contents

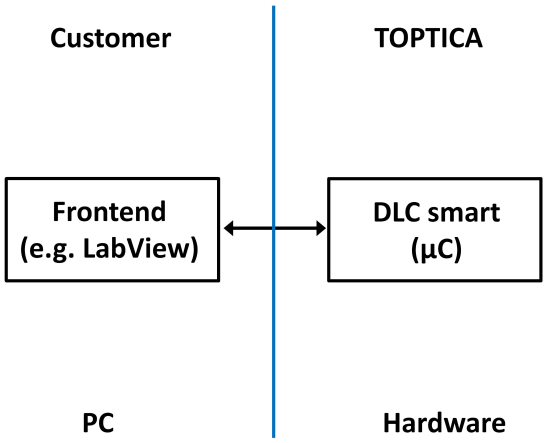

Figure 1: Software architecture.

1 Introduction

Running the software of TOPTICA's TeraScan / TeraBeam systems requires an external laptop or PC with Windows XP / Windows Vista / Windows 7 / Windows 8.

The software architecture comprises two levels (Fig. 1). On the hardware side, the DLC smart features a powerful microprocessor, which sets all relevant laser parameters and processes the terahertz signals received. The DLC smart provides an Ethernet port for remote control. On the frontend side, TOPTICA provides a professional graphical user interface ("TOPAS_TeraScanControl.exe"). Alternatively, customers may choose to write their own frontend software, e.g. in LabView or C++. In the latter scenario, the TeraScan / TeraBeam system is controlled via a set of software commands. This option is recommended for advanced experimental setups that comprise further computer-controlled components, e.g. a raster scanning stage for THz imaging.

Once the TOPAS_TeraScanControl.exe frontend is started, users can select any DLC smart within their network, and open a TCP/IP connection for remote control of a specific device. Similarly, if customers wish to use their own software frontend, it needs to open a TCP/IP connection to communicate with the DLC smart.

1.1 Ethernet Connection

The DLC smart is addressed via Ethernet. You can either connect the system to a local area network (LAN) or directly to your computer. In both cases, the TCP/IP network address of the device has to be configured correctly. Specifically, both the DLC smart and the control computer need to have IP addresses of the same subnet. In many LANs the configuration can be done automatically by a DHCP server.

The DLC smart listens on three different ports. Upon delivery,

- port 1998 is used for the command line,

- port 1999 is used for the monitoring line,
- port 60010 is used for the identification line.

1.1.1 Command Line — Telnet Server / Scheme Console

A Telnet server on standard port 1998 provides access to the command console. This is the connection intended for automated remote control via Ethernet. On this console, you can use the complete set of control commands as listed in the following sections.

Since all instructions and their responses consist only of ASCII characters, you can use any standard Telnet client. Windows provides e.g. Hyperterminal or the Telnet command at the MSDOS console. You can also try entering `telnet://xxx.xxx.xxx.xxx` into the address bar of your web browser to let Windows choose its favorite Telnet client. Another good choice is *PuTTY*[1]. The Telnet command also belongs to every standard Linux distribution.

For system control within your own software framework, please follow these steps:

1. Open a TCP/IP connection to port 1998 at the IP address of the DLC smart.
2. The system then sends a welcome text which is completed by a prompt consisting of the two characters `>␣` ("greater than" and space) at the beginning of a new line.
3. Send your commands and complete every instruction with a linefeed character (ASCII code 10 decimal, 0x0A hexadecimal).
4. The reply of the system is once again completed by the prompt.
5. When finished, close the TCP/IP connection or send `(quit)`.

The DLC smart accepts up to five Telnet connections at the same time. This is useful e.g. for applications where one GUI software takes control of the DLC smart while a second connection serves for debugging.

1.1.2 Monitoring Line

The monitoring line enables advanced automatic surveillance of device parameters. This feature can be useful for manual debugging via the console or synchronization of a customized frontend software with the DLC smart, without any need for continuous polling. To use the monitoring line, a Telnet connection on port 1999 of the DLC smart has to be established.

[1] PuTTY is a free Telnet and SSH client for Windows and UNIX platforms. It can be downloaded from the official site http://www.chiark.greenend.org.uk/~sgtatham/putty/

1.1.3 Identification Line

The firmware of the DLC smart is based on DeCoF, TOPTICA's proprietary "Device Control Framework". In order to determine the IP address of the DLC smart (or any other TOPTICA DeCoF device), the DeCoF devices provide a UDP-based identification service on port 60010.

If you want to program a software search to find a particular system within the LAN, proceed as follows:

1. On port 60010, broadcast a short string like "laserfinder" to the entire LAN (address 255.255.255.255) or to the broadcast address of your network adapter (e.g. 192.168.1.255).

2. Listen for an answer on the same port. The system will reply with a UDP message providing information about its serial number and its firmware version. The IP address of the device is comprised in the header of the UDP packet.

If no message is received within one second, no DeCoF device is available. If more than one system is connected to the LAN, this procedure returns a list of all devices.

2 Control Language

The software control of the DLC smart is based on a programming language called *Scheme*.

Scheme instructions always have the same simple syntax. They are enclosed by parentheses and use prefix notation. That means the function/instruction name is always the first text inside the parentheses. If the instruction requires arguments, they are separated by whitespaces (spaces, tab or linefeed):

```
(cmd arg1 arg2)
```

Terminate your instructions with a linefeed character (ASCII code 10 decimal, 0xA hexdecimal). Execution is finished when you receive a *prompt*. The prompt consists of the characters `>␣` ("greater than" and space) at the beginning of a new line.

Every instruction returns a value (e.g. integer, string, boolean) that can be used by other instructions or can be assigned to a variable. The return value is always the last item before the prompt.

Example:

```
> (+ 13 7 5)
25
>
```

Instructions can be nested, i.e. each element of an instruction can be the result of another instruction:

```
> (+ 13 (/ 21 3) (* 2.5 2))
25
>
```

Errors are indicated by the string "`Error: `" at the beginning of a new line:

```
> (/ 1 0)
Error: -28 /: division by zero
>
```

Some instructions print out additional text before the result value:

```
> (display "Hello World\n")
Hello World
#t
>
```

Here the boolean value `#t` is the return value which can be used in nested instructions, whereas the string "Hello World" is just a print-out.

2.1 Parameters

TOPTICA's DeCoF systems such as the DLC smart are controlled by both *parameters* and *commands*.

- Parameters represent certain properties of the device, e.g. a photocurrent value, a serial number or an enable/disable flag etc.
- Commands are used to execute certain tasks like starting a recalibration or starting a scan. However, commands can also be used instead of a parameter, e.g. to obtain data as a function of some arguments.

Each parameter has a certain name, a certain type and a certain mode.

Parameter names start with a letter and may contain letters, numerical digits and special characters like hyphens, underscores etc.

The *parameter type* can be one of the following:

integer - integer numbers, e.g. -10, 0 or 1998

real - floating-point numbers, using decimal points, e.g. 3.1415 or 6.625e-34

string - text enclosed in quotation marks, e.g. `"TOPTICA Photonics"`

boolean - true/false values, `#t` = true, `#f` = false

binary - an ASCII string which represents Base64-encoded data. This data type is typically used to transfer large data arrays. For details see Appendix 4.2.

tuples - a set of several values of the above types in a fixed order, enclosed by parentheses, e.g. a pair of XY coordinates: `(100 200)`.

The *parameter mode* comprises information about the way it can be accessed:

read-only - these parameters cannot be modified but can only be readout.

read-write - these parameters can be modified. The new value will immediately take effect and will be returned when the parameter is readout.

read-set - these parameters can be modified. Assigning a new value to such a parameter, however, will only change the target value or setpoint. An example is the set temperature of a temperature-stabilization loop. The DLC smart does however not use any read-set parameters.

There are four different *Scheme instructions* for accessing parameters: `param-ref`, `param-set!`, `param-gsv` and `param-disp`.

2.1.1 param-ref

The value of a parameter can be retrieved with the *param-ref* instruction. The return value is the value of the parameter being queried. Examples:

```
> (param-ref 'int-param)
8375309
> (param-ref 'str-param)
"this is a string"
> (param-ref 'bool-param)
#t
> (param-ref 'real-param)
3.141592
> (param-ref 'tuple-param)
(0.8 15)
>
```

Note that parameter names have to be quoted with a single, straight quotation mark. Reading out parameter values is possible for parameters of any mode.

2.1.2 param-set!

The *param-set!* instruction changes the value of a parameter. Its return value is zero or a positive integer, the latter indicating a warning. Here are a few examples for parameters of different types:

```
> (param-set! 'label "Example Device")
0
> (param-set! 'int-param 50)
0
> (param-set! 'bool-param #f)
0
> (param-set! 'tuple-param '(15 0.8))
0
>
```

Modifying a parameter value is only possible for *read-write* and *read-set* parameter modes. Tuple-type arguments have to be addressed with a single, straight quotation mark.

Please remember that nesting of instructions is possible. The following example shows a way to toggle a boolean parameter:

```
> (param-set! 'bool-param (not (param-ref 'bool-param)))
0
>
```

2.1.3 param-gsv

Read-write-mode parameters comprise two values that can be readout: the actual value and the setpoint value. Whilst the *param-ref* instruction returns the actual value, you can use *param-gsv* to retrieve the setpoint value:

```
> (param-ref 'temp)
24.9128
> (param-gsv 'temp)
25
>
```

2.1.4 param-disp

Whilst the *param-ref* instruction is the preferred way for software-implemented parameter queries, there is a more easily readable alternative for manual queries on the command console: the *param-disp* instruction. In contrast to the *param-ref* instruction, *param-disp* displays the parameter values together with their names while its return value is always zero.

Generally, *param-disp* instructions can be used with single parameters, but are most useful in conjunction with parameter sections:

```
> (param-disp 'laser1)
laser1
  :label = "640 nm"
  :type = "640nm laser diode"
0
>
```

You can use the `(param-disp)` instruction without a parameter name to get a complete list of available parameters.

2.2 Commands

Commands are called by the *exec* instruction, like in the following example:

```
> (exec 'hello)
()
>
```

Note that command names, just like parameter names, have to be quoted by a single, straight quotation mark. Like in the example, commands often do not return any specific result value but an empty tuple `()` instead[2]. Some commands require one or more additional arguments, which are included in the parentheses and separated by whitespaces.

```
> (exec 'frequency:fast-scan-start)
0
> (exec 'frequency:fast-scan-get-data 1 0 1024)
""
> (exec 'net-conf:set-ip "192.168.1.1" "255.255.255.0")
()
```

2.3 Errors and Warnings

Errors are indicated by the string "`Error: `" followed by a negative-number error code and a brief error description:

```
> (param-ref this-parameter-does-not-exist)
Error: -3 no such parameter
> (param-set! 'general:fw-ver "new text")
Error: -11 parameter not settable
> (param-set! 'general:serial-number "new text")
Error: -22 no access
> (param-set! 'general:system-label 123)
Error: -1 invalid argument
>
```

A positive number returned by *param-set!* indicates a warning. For some numerical parameters, a return value of 2 indicates *"parameter value clipped"*. That is, the specified value has exceeded the valid data range and the parameter was set to the minimum or maximum allowed value. Use *param-ref* to retrieve the value actually set by the device. Example:

```
> (param-set! 'laser-operation:laser1:current-set 5000)
2
> (param-ref 'laser-operation:laser1:current-set)
225
>
```

2.4 Remote Monitoring

To use the remote monitoring feature, a Telnet connection on the monitoring line port of the DLC smart has to be established (see section 1.1.2). The syntax of the monitoring line is similar to that of the Scheme commands, see Sec. 2. The structure of a monitoring command is:

[2] All Scheme instructions must return a value.

```
(cmd_name 'param_name [period] [threshold])
```

cmd_name Possible commands are:

add Adds a specific parameter to the monitoring line, e.g. `(add 'tec:temp 100 0.1)`

remove Removes a specific parameter from monitoring, e.g. `(remove 'laser1:beampos)`

remove-all Removes all parameters from the monitoring line: `(remove-all)`

query Reads a momentary parameter value, e.g. `(query 'laser1:beampos)`

param_name The name of the parameter to be monitored.

period Update interval in ms. After every period, an updated value is sent, provided that the parameter has changed. The change must be above the "threshold" value.

threshold A threshold value for sending updates. If a parameter changes by less than the threshold, no update will be sent.

If an error occurs, an empty list `()` is returned. Otherwise the current value of the registered parameter is returned. A value looks like:

```
(timestamp 'param_name [set_value!] value)
```

Example:

```
(102393494 'laser1:beampos (23 21)! (10 23))
(102393598 'laser1:on #t)
```

timestamp A time stamp for a specific value. The definition of the time stamp depends on the host system; usually, the time stamp has the unit of milliseconds.

param_name The name of the monitored parameter.

set_value The set-value of the parameter. The set-value is marked with an exclamation mark `!`. The set-value is optional.

value The actual value of the parameter at the time given by "timestamp".

3 List of all Parameters and Commands

general:

(parameter section)

General parameters and commands of the DLC smart.

general:serial-number

(STRING parameter, read-only)

If connected to a DLC smart, the returned string contains the serial number of the device, e.g. “DLC smart_010076”.

general:system-type

(STRING parameter, read-only)

If connected to a DLC smart, the returned string must read “DLCsmartTHz”. This command can be used to test the communication with the DLC smart.

general:system-label

(STRING parameter, read-write)

A configurable string for easy identification of the device, like “TOPTICA spectrometer (THz lab)”. Upon delivery this string contains the serial number of the device, e.g. “DLC smart_010076”.

general:system-manufacturer

(STRING parameter, read-only)

This string identifies the manufacturer of the system, i.e. “TOPTICA Photonics”.

general:fw-ver

(STRING parameter, read-only)

Returns the firmware version of the DLC smart.

general:profiles:

(parameter section)

Parameters and commands for different system profiles. These parameters are especially useful for systems with Tuning Range Extension which feature a third laser head.

general:profiles:active-profile

(STRING parameter, read-only)

Returns the name of the active profile. The name consists of the serial numbers of the laser heads, e.g. “33418-33419”. For legacy laser heads not featuring an EEPROM containing the serial number, the active profile name is “Legacy”. If no valid combination of laser heads is detected, a safety profile is loaded and the returned string reads “Safety”.

general:profiles:profile-caption

(STRING parameter, read-only)

Provides an additional caption for the **:active-profile**, e.g. “THz: -50 .. 1300 GHz”.

general:legal-info

(command, no arguments, returns empty tuple)

Prints a (base64-encoded) data stream which contains license and copyright information associated with third-party software incorporated in the DLC smart.

laser-operation:

(parameter section)

Control parameters for the DFB lasers of the TeraScan / TeraBeam system.

laser-operation:interlock-open

(BOOLEAN parameter, read-only)

Returns `#t` if the external interlock circuit is open, preventing laser emission. If the external interlock is closed, the return value is `#f`.

laser-operation:frontkey-locked

(BOOLEAN parameter, read-only)

Returns `#f` if the key switch on the front panel of the DLC smart is unlocked, otherwise `#t`. Laser emission is only possible if the key switch is unlocked.

laser-operation:emission-global-enable

(BOOLEAN parameter, read-write)

Global parameter to enable (`#t`) or disable (`#f`) the laser emission. This parameter is also set by the hardware buttons “Emission On” (green) and “Emission Off” (red).

laser-operation:emission

(BOOLEAN parameter, read-only)

Returns `#t` if laser emission is possible. If the return value is `#f`, then laser emission has been disabled, either by the **:emission-global-enable** parameter, or by the **:laser1:emission-enable** and **:laser2:emission-enable** parameters, or by the hardware (key switch, interlock, or the red “Emission Off” button on the front panel).

laser-operation:laser1:

(parameter section)

Parameters to control the operation of laser1.

laser-operation:laser1:serial-number

(STRING parameter, read-only)

Returns the serial number of the laser head, e.g. "33419". For legacy laser heads not featuring an EEPROM containing the serial number, "unknown" is returned.

laser-operation:laser1:emission-enable

(BOOLEAN parameter, read-write)

This parameter controls the emission of each laser head individually. Setting **:emission-enable** to `#t` enables the operation of this specific laser; setting it to `#f` stops and disables laser emission. See also **laser-operation:emission-global-enable**.

laser-operation:laser1:emission

(BOOLEAN parameter, read-only)

Returns `#t` if laser emission is possible. If the return value is `#f`, then laser emission has been disabled, either by the **:emission-global-enable** parameter, or by the **:emission-enable** parameter of this specific laser, or by the hardware (key switch, interlock, or the red "Emission Off" button on the front panel).

laser-operation:laser2:

(parameter section)

Parameters to control the operation of laser2.

For usage see **laser-operation:laser1:** on page 16.

frequency:

(parameter section)

Parameters and commands to control the terahertz frequency.

The TeraScan / TeraBeam systems offer two different scan modes, denoted "Precise Scan" and "Fast Scan". The respective mode is chosen by **:scan-mode-fast**.

In Precise mode (**:scan-mode-fast** = `#f`), the terahertz frequency is controlled by the parameter **:frequency-set** (value of the terahertz frequency, in GHz). In addition, **:frequency-act** can be used to retrieve the actual terahertz frequency as calculated from the measured, instantaneous laser temperatures. In Precise mode, other parameters such as **:frequency-min**, **:frequency-max**, **:frequency-step** etc. are not used. A frequency scan has to be composed of individual frequency steps, implemented by

the client application (e.g. the GUI). This mode offers maximal flexibility, but the scan speed is limited by the communication data rate.

The Fast Scan mode (**:scan-mode-fast** = `#t`) is implemented within the device. The parameters **:frequency-min**, **:frequency-max**, and **:frequency-step** define the lower and upper limit as well as the step size of the frequency scan. The command **:fast-scan-start** initiates a scan.

Note that a scan always starts from the instantaneous set frequency (**:frequency-set)**. The sign of **:frequency-step** defines the direction of the scan. As long as the scan is running, **:fast-scan-isscanning** is `#t`. The scan terminates once the limit of the scan range, i.e. **:frequency-min** or **:frequency-max**, is reached. Alternatively, the scan can be stopped any time by **:fast-scan-stop**. Data sampled during the scan can be readout—either at the end of the scan or while the scan is still running—via **:fast-scan-get-data**.

frequency:scan-mode-fast

(BOOLEAN parameter, read-write)

`#t` for Fast Scan. `#f` for Precise Scan.

frequency:frequency-set

(REAL parameter, read-write)

Terahertz set frequency in GHz.

frequency:frequency-min

(REAL parameter, read-write)

Minimum frequency of a scan, in GHz.

frequency:frequency-max

(REAL parameter, read-write)

Maximum frequency of a scan, in GHz.

frequency:frequency-step

(REAL parameter, read-write)

Size of an individual frequency step of a scan, in GHz. A positive step size defines a scan from lower to higher frequencies. A negative step size defines a scan from higher to lower frequencies.

frequency:frequency-act

(REAL parameter, read-only)

Actual terahertz frequency in GHz as calculated from the measured, instantaneous laser temperatures.

frequency:fast-scan-isscanning

(BOOLEAN parameter, read-only)

`#t` as long as the Fast Scan is running.

frequency:fast-scan-start

(command, no arguments, returns INTEGER)

Starts a Fast Scan.

frequency:fast-scan-stop

(command, no arguments, returns INTEGER)

Stops a Fast Scan.

frequency:fast-scan-clear-data

(command, no arguments, returns INTEGER)

Clears the data of the last or the active scan.

frequency:fast-scan-get-data

(command, 3 arguments, returns BINARY)

Arguments:

1. *curveNo* of type INTEGER
2. *startIndex* of type INTEGER
3. *length* of type INTEGER

Retrieves the raw data (i.e. point number, set frequency, act. frequency and photocurrent values) of a scan. The returned value is of type "Binary" (cf. Appendix 4.2), Base64-encoded. After decoding, it comprises *length* double values, i.e. $8 \times$*length* bytes.

Internally, the DLC smart stores all the data collected during a Fast Scan in a $7 \times n$-array (7 curves, each with an arbitrary number n of data points). However, you can only readout up to 1024 data points (double values) at a time. Therefore, **:fast-scan-get-data** has to be called repeatedly until all data is readout.

Curve numbers 0, 1, 2 and 6 refer to the point number, terahertz set frequency, terahertz photocurrent, and terahertz act. frequency, respectively:

curveNo	Meaning
0	Point Number
1	Terahertz set frequency (GHz)
2	Terahertz photocurrent (nA)
3	not implemented
4	not implemented
5	not implemented
6	Terahertz act. frequency (GHz)

For each curve, *length* data points (double values) starting from *startIndex* are returned.

If *length* is smaller than 0 or greater than 1024, it is set to 1024. If less than *startIndex* + *length* data points are available, all data points starting from *startIndex* are returned.

lockin:

(parameter section)

Parameters and commands for lock-in detection of the terahertz photocurrent.

The DLC smart offers two different lock-in modes, with and without fast phase modulation. The respective mode is chosen by **:fast-phase-modulation:enabled** (Checkbox "Fast Phase Modulation via Fiber Strecher" in the TOPAS GUI).

In the mode without fast phase modulation (**:fast-phase-modulation:enabled** = `#f`), the Tx bias voltage is modulated at a high frequency (several kHz, see **:frequency**). A digital lock-in amplifier demodulates the Rx signal at the same frequency and yields the terahertz photocurrent. To retrieve the terahertz photocurrent proceed as follows:

1. Reset the lock-in integration (**:lock-in-reset**).
2. Wait for one integration time (**:integration-time**).
3. Readout the lock-in output value, i.e. the terahertz photocurrent (**:lock-in-value**).

On the other hand, the fast-phase-modulation mode (**:fast-phase-modulation:enabled** = `#t`) activates a pair of fiber stretchers. In this case, the phase modulator voltage is modulated at a moderate frequency, e.g. at 800 Hz (see **:fast-phase-modulation:frequency**). The Rx photocurrent is averaged by a digital signal averager. To acquire terahertz data,

1. Reset/Start the signal averager (**:fast-phase-modulation:clear**).
2. Wait as long as the averaging shall last, e.g. 10 .. 128 ms. (Note: For averaging times longer than 128 ms and large input signals, the signal averager can overflow without notice.)
3. Stop the signal averaging (**:fast-phase-modulation:stop-acquisition**).
4. Wait until the data is valid (**:fast-phase-modulation:data-valid**).
5. Readout the data, either the sinusoidal raw-data curves (e.g. **:fast-phase-modulation:raw-data-monitored-phase-volt** and **:fast-phase-modulation:raw-data-photocurrent-volt**) or the amplitude and phase values as determined from a least-squares fit (**:fast-phase-modulation:fit-data-amplitude-volt** and **:fast-phase-modulation:fit-data-phase**).

A graphical representation of these steps is given in the flowcharts in Appendix 4.1.

lockin:fast-phase-modulation:

(parameter section)

Parameters and commands for the fast phase modulation.

lockin:fast-phase-modulation:enabled

(BOOLEAN parameter, read-write)

If `#t`, the fiber stretcher (fast phase modulator) is activated and the signal averager is used instead of the standard lock-in amplifier.

lockin:fast-phase-modulation:data-valid

(BOOLEAN parameter, read-only)

If `#t`, the data acquisition is finished and the data are valid. If `#f`, the data acquisition is still ongoing and the readout data may contain inconsistent values.

lockin:fast-phase-modulation:raw-data-monitored-phase-volt

(BINARY parameter, read-only)

Monitor signal of the fiber stretcher, in Volt. The parameter represents a time-dependent voltage, proportional to the output signal of the HV amplifier that drives the stretchers. Data are signal-averaged. The returned value is of type "Binary" (cf. Appendix 4.2), Base64-encoded. After decoding, it comprises 8192 bytes, i.e. 1024 double values of 8 bytes each.

lockin:fast-phase-modulation:raw-data-photocurrent-volt

(BINARY parameter, read-only)

Receiver photocurrent, in units of Volt, at the DLC smart input. Data are signal-averaged. The returned value is of type "Binary" (cf. Appendix 4.2), Base64-encoded. After decoding, it comprises 8192 bytes, i.e. 1024 double values of 8 bytes each.

lockin:fast-phase-modulation:raw-data-photocurrent-nanoamp

(BINARY parameter, read-only)

Receiver photocurrent in nA. Data are signal-averaged. The returned value is of type "Binary" (cf. Appendix 4.2), Base64-encoded. After decoding, it comprises 8192 bytes, i.e. 1024 double values of 8 bytes each.

lockin:fast-phase-modulation:fit-data

(TUPLEFITDATA parameter, read-only)
Type TUPLEFITDATA is defined as a tuple (a0 a1 a2 a3 chi2 fitok) of types (REAL REAL REAL REAL REAL BOOLEAN).

Performs a cosine fit of the raw data and yields the fit results.
The fit function is

$$y(x) = a_0 \cos(a_1 x + a_2) + a_3$$

where y is the Rx photocurrent in Volt at the DLC smart input and x is the monitor voltage (in Volt), proportional to the path-length modulation of the stretchers. The returned tuple contains 5 real values and one boolean value:

index	type	meaning
0	real	amplitude a_0
1	real	scaling factor a_1
2	real	phase a_2
3	real	offset a_3
4	real	sum of squared residuals χ^2
5	bool	fit ok?

$\chi^2 = \sum_i (y(x_i) - y_i)^2$ where (x_i, y_i) are the measured data points.

Since y is in Volt, a_0 and a_3 are also in Volt. Use **:amplifier-gain** to calculate the photocurrent in nA.

lockin:fast-phase-modulation:fit-data-amplitude-volt

(REAL parameter, read-only)

Calls **:fit-data** and returns the fit parameter a_0, i.e. the photocurrent amplitude in Volt. Use **:amplifier-gain** to calculate the photocurrent in nA or call **:fit-data-amplitude-nanoamp**.

lockin:fast-phase-modulation:fit-data-amplitude-nanoamp

(REAL parameter, read-only)

Same as **:fit-data-amplitude-volt**, but the photocurrent amplitude is given in units of nA, calculated with the help of **lockin:amplifier-gain**.

lockin:fast-phase-modulation:fit-data-phase

(REAL parameter, read-only)

Calls **:fit-data** and returns the fit parameter a_2, i.e. the phase in rad.

lockin:fast-phase-modulation:fit-data-chi-square

(REAL parameter, read-only)

Calls **:fit-data** and returns χ^2.

lockin:fast-phase-modulation:clear

(command, no arguments, returns INTEGER)

Clears/resets the signal averager.

lockin:fast-phase-modulation:stop-acquisition

(command, no arguments, returns INTEGER)

Stops the data acquisition of the signal averager.

lockin:frequency

(REAL parameter, read-write)

Modulation frequency f_{lockin} of the Tx bias amplitude, in Hz. Settings for a specific TeraScan system are documented in the *Production and Quality Control Datasheet*, Section 05.

lockin:phase

(REAL parameter, read-write)

Reference phase for the lock-in detection, in degrees. The easiest way to adjust the lock-in phase is, to vary the value until the terahertz photocurrent equals zero, and then add or subtract 90 degrees.

lockin:integration-time

(REAL parameter, read-write)

Lock-in integration time t_{int} per frequency data point, in milliseconds. See also **:integration-time-matches-modulation-frequency**.

lockin:integration-time-must-match-modulation-frequency

(BOOLEAN parameter, read-write)

If `#t`, the integration time t_{int} is always rounded to the next integer multiple of the modulation period, i.e. the inverse modulation frequency f_{lockin} (see **:frequency**):

$$t_{\text{int}} = n/f_{\text{lockin}}, \quad n \in N.$$

lockin:integration-time-matches-modulation-frequency

(BOOLEAN parameter, read-only)

`#t`, if the integration time t_{int} is an integer multiple of the modulation period, i.e. the inverse modulation frequency f_{lockin} (see **:frequency**):

$$t_{\text{int}} = n/f_{\text{lockin}}, \quad n \in N.$$

lockin:integration-time-step

(REAL parameter, read-only)

Minimum allowed step of the integration time in ms, if **:integration-time-must-match-modulation-frequency** is `#t`. If no valid integration time greater than zero exists, a negative value is returned. See also **:integration-time**.

lockin:mod-out-amplitude

(REAL parameter, read-write)

AC modulation amplitude for the transmitter photomixer. The signal is available at an SMB-connector labelled "Bias Out" at the rear panel of the DLC smart (either output 1 (2.5 V) or 2 (12.5 V), as documented in the *Acceptance Report*). The signal is sinusoidal in the configuration without fast phase modulation and rectangular in systems with fast phase modulation. In either case, the signal is symmetric with respect to the adjusted **:mod-out-offset**.

lockin:mod-out-offset

(REAL parameter, read-write)

DC output for the transmitter photomixer. The signal is available at an SMB-connector labelled "Bias Out" at the rear panel of the DLC smart (either output 1 (2.5 V) or 2 (12.5 V), as documented in the *Acceptance Report*).

lockin:mod-out-amplitude-default

(REAL parameter, read-only)

Default value for **:mod-out-amplitude**.

lockin:mod-out-offset-default

(REAL parameter, read-only)

Default value for **:mod-out-offset**.

lockin:lock-in-value

(TUPLEREALBOOLEAN parameter, read-only)
Type TUPLEREALBOOLEAN is defined as a tuple (value-real value-boolean) of types (REAL BOOLEAN).

To readout the photocurrent, first use **:lock-in-reset**. After that, wait for a time interval corresponding to the lock-in integration time. The returned boolean value indicates whether the integration has been completed. If `#t`, then the real value returned is the de-modulated output of the lock-in amplifier. It represents the photocurrent in the terahertz receiver, in units of Volt, at the DLC smart input. Use **:lock-in-value-nanoamp** to obtain the photocurrent in units of nA.

lockin:lock-in-value-nanoamp

(TUPLEREALBOOLEAN parameter, read-only)
Type TUPLEREALBOOLEAN is defined as a tuple (value-real value-boolean) of types (REAL BOOLEAN).

Same as **:lock-in-value**, but the photocurrent is given in units of nA, calculated with the help of **:amplifier-gain**.

lockin:amplifier-gain

(REAL parameter, read-write)

Gain of the external transimpedance amplifier (in V/A). This value is only used to calculate the receiver photocurrent (in nA). It does not control the external amplifier device. If **:amplifier-gain** is set to 1, the calculated values equal the voltage at the DLC smart input (in nV). For the TOPTICA PDA-S the available gain levels are (with the "var. gain" potentiometer set to 100%):

gain setting	gain (V/A)
1	$3.3 * 10^4$
2	10^5
3	$3.3 * 10^5$
4	10^6
5	$3.3 * 10^6$
6	10^7

lockin:mod-out-set-to-default

(command, no arguments, returns INTEGER)

Sets **:mod-out-offset** to **:mod-out-offset-default** and **:mod-out-amplitude** to **:mod-out-amplitude-default**.

lockin:mod-out-set-to-zero

(command, no arguments, returns INTEGER)

Sets **:mod-out-offset** and **:mod-out-amplitude** to zero.

lockin:lock-in-reset

(command, no arguments, returns INTEGER)

Resets the lock-in integration. After reset, the (photocurrent) value read by **:lock-in-value** is invalid for the duration of the lock-in integration time. Once the integration is completed, the value is held until the next **:lock-in-reset** is executed.

net-conf:

(parameter section)

The DLC smart is addressed via Ethernet. You can either connect it to a local area network (LAN) or directly to your computer. In both cases, the TCP/IP network address of the DLC smart has to be configured correctly. In many LANs the configuration can be done automatically by a DHCP server. For a direct PC-to-device connection, the device usually needs to obtain a fixed network address. The parameters and commands of this section provide information about the default network configuration and how to change it.

net-conf:ip-addr

(STRING parameter, read-only)

Returns the current IP address of the DLC smart[3]. To change the IP address, use the **net-conf:set-dhcp** command for automatic address retrieval in a LAN with DHCP server, or **net-conf:set-ip** for manual configuration.

net-conf:net-mask

(STRING parameter, read-only)

Returns the current netmask of the DLC smart. To change the netmask, use the **net-conf:dhcp** command for automatic address retrieval in a LAN with DHCP server, or **net-conf:set-ip** for manual configuration.

net-conf:mac-addr

(STRING parameter, read-only)

Returns the MAC address of the DLC smart.

net-conf:dhcp

(BOOLEAN parameter, read-only)

Indicates whether the DLC smart is configured for automatic IP address retrieval by a DHCP server.

net-conf:cmd-port

(INTEGER parameter, read-only)

Returns the TCP/IP port which provides access to the command line console.

net-conf:mon-port

(INTEGER parameter, read-only)

Returns the TCP/IP port which provides access to the monitoring line of the DLC smart. If this parameter returns 0, then monitoring lines are not supported by this system.

net-conf:set-dhcp

(command, no arguments, returns empty tuple)

Changes the network adapter configuration for automatic setup by a DHCP server. After issuing this command, the parameter **net-conf:dhcp** returns `#t`, but IP address and netmask have not changed yet. The new network adapter settings will take effect upon the next adapter restart. In order to use the DHCP mechanism, reboot the system or use the **net-conf:apply** command.

[3] If you want to access your DLC smart via TCP/IP but don't know the current IP address (e.g. in case of DHCP configuration), you can use the "Identification Server" described in section 1.1.3 to find out.

net-conf:set-ip

(command, 2 arguments, returns empty tuple)

Arguments:

1. *ip-addr* of type STRING
2. *net-mask* of type STRING

Changes the network adapter configuration to a static IP configuration. Both the desired IP address and netmask have to be provided as IPv4 addresses in the format "xxx.xxx.xxx.xxx", e.g.:

```
(exec 'net-conf:set-ip "192.168.1.1" "255.255.255.0")
```

After issuing this command, the parameter **net-conf:dhcp** will return `#f`, but IP address and netmask have not changed yet. The new network adapter settings will take effect upon the next adapter restart. In order to use the new address, reboot the system or use the **net-conf:apply** command.

net-conf:apply

(command, no arguments, returns empty tuple)

Restarts the network adapter, making use of the current network configuration. Please note that your current TCP/IP connections might be lost if you use this command.

ul

(INTEGER parameter, read-write)

Returns a number representing the currently active user level.

Some parameters and commands are not made available to all users, either because they are not required for regular operation, or because they can adversely affect the system performance. The following user levels are implemented:

4 - read-only level, lowest priority

3 - normal operation

2 - maintenance level

1 - service level

0 - TOPTICA internal

The lower the user level the more parameters and commands are accessible. It is always possible to change the **ul** parameter to a higher value, but setting it to a lower value requires the **change-ul** command.

change-ul

(command, 2 arguments, returns INTEGER)

Arguments:

1. *ul* of type INTEGER
2. *passwd* of type STRING

Changes the user level.

All user levels except for the read-only level (4) are password protected. The password for normal operation (user level 3) can be configured with the **change-password** command. By default, it is an empty string. The higher-priority user-level passwords are predefined.

In order to change the user level, enter the desired level as first argument, and the corresponding password as second argument. Passwords of higher-priority levels are valid for all lower-priority levels as well.

If you provide an empty string instead of the correct password as second argument, the system prints `password:` to prompt you for input. You may then enter the password without any letters being displayed.

The return value is the new user level.

change-password

(command, 1 argument, returns empty tuple)

Arguments:

1. *password* of type STRING

Command to change the password of user level 3 (normal operation). If the password is not empty, any subsequent remote connection starts in user level 4 (read-only level) and **change-ul** must be called to change to user level 3 (normal operation). It is not possible to readout the password.

4 Appendix

4.1 Flowcharts

The flowcharts in Fig. 2 and Fig. 3 show possible implementations of the measurement loop of a step-wise frequency scan. Fig. 2 describes a very simple program which scans the frequency and reads the terahertz photocurrent. Fig. 3 describes a more sophisticated program including the fast phase modulation.

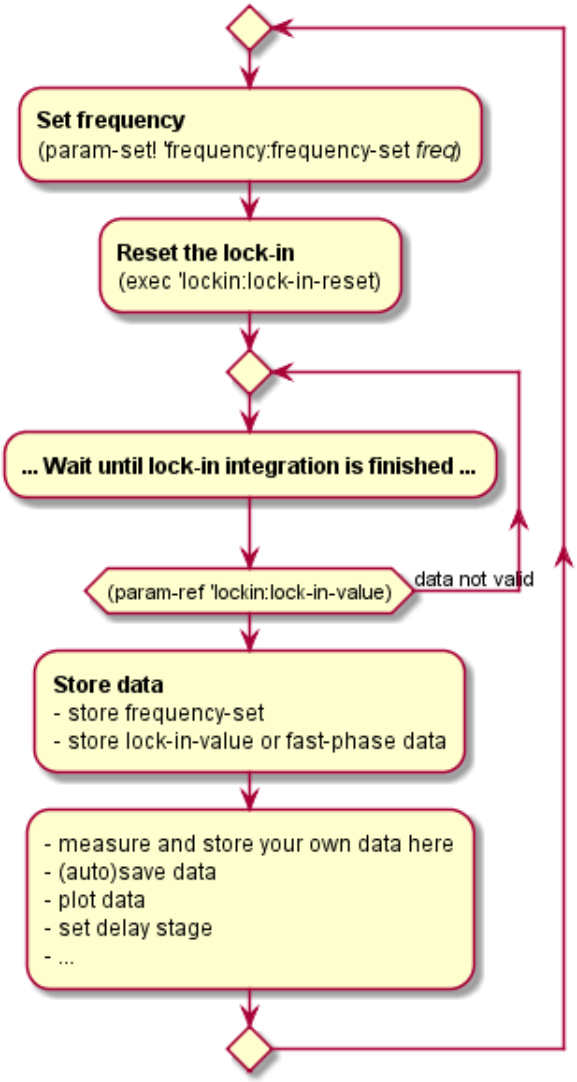

Figure 2: Flowchart of a possible implementation of a step-scan measurement-loop.

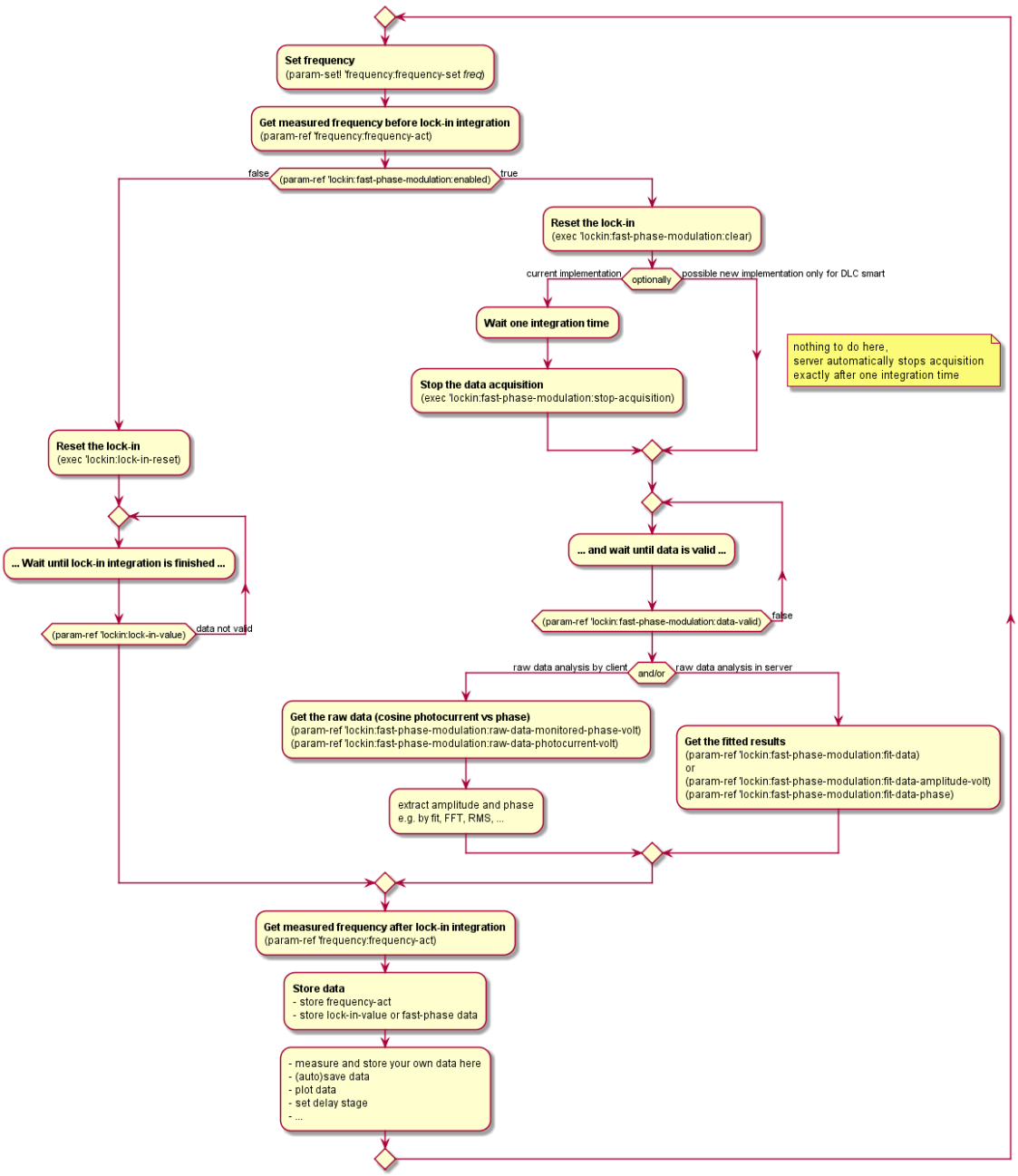

Figure 3: Flowchart of a possible implementation of a step-scan measurement-loop including the fast phase modulation.

4.2 Binary Data

The data type "Binary" is used to transfer large data arrays in parameters like **lockin:fast-phase-modulation:raw-data-photocurrent-volt** or **frequency:fast-scan-get-data**. The DeCoF protocol is a pure ASCII protocol. Therefore, binary data is not transmitted as is, but Base64-encoded. Base64 data uses only 6 bit per byte. A binary array of n bytes therefore results in $n \times 8/6$ bytes of Base64 code. The following alphabet is used for Base64 encoding and decoding:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/
```

To interpret the return value of a "Binary"-type parameter, the first step is to remove the enclosing quotation marks and to decode the Base64 data into a binary data buffer. If you use the Qt library, there is a function QByteArray::fromBase64() available.

4.2.1 Floating Point Format

Often numerical values are represented as double precision floating point values with 64 bit, according to IEEE 754 standard. The most significant bit 63 contains the sign, bits 52 to 62 contain the exponent, and bits 0 to 51 contain the mantissa.

63	62 52	51 0
±	11 bit exponent	52 bit mantissa

These 64 bits are transferred in little endian manner, starting with the least significant byte.

first byte		second byte		...	7th byte		8th byte	
7	0	15	8	...	55	48	63	56

This is in accordance with the native way of storing double precision floating point numbers on x86-based computers and, for example, the *double* data type in C on such computers. Please note that LabVIEW, in contrast to most other programming languages, uses big endian format, independent of the computer hardware.

If arrays of *double* values are transferred, then the length of the decoded binary data equals the length of the *double* array multiplied by 8 (size of *double*). In the decoded binary data, blocks of 8 bytes can be assigned (type cast) to values of type double.

Index