XL3 API Manual

17 April, 2026

NTi Audio, Im alten Riet 102, 9494 Schaan, Liechtenstein

Preamble

This document refers to XL3 firmware version **1.54** or higher

The XL3 is designed for easy integration into Noise Monitoring and measurement applications and offers the following services:

Service	Description	Application
Streaming API	**Basic Streaming API** Pushes the values seen on the 123 screen and the bar spectrum to up to 20 clients.	Dashboards
	Advanced Streaming API Stream SPL (Sound Pressure Level), SOH (State of Health), Weather and Audio for easy integration into Noise Monitoring applications.	Noise Monitoring
Control API	Control and manually query SLM, RT and STIPA measurements.	Test Systems
File Push Service	Push measurement results to Google Drive, OneDrive, WebDAV or SFTP clients.	Various
SFTP	File access via an SSH connection (not to be confused with FTPS).	Special cases

API (Programming Interface) Option required
You need the API (Programming Interface) Option installed on your XL3 to include
- the Advanced Streaming and
- the Control API

interface in your application.

The Basic Streaming API, File Push Service and SFTP access are available by default. Refer to the XL3 User Manual for further details.

Quick Start

1. Connect the XL3 with the supplied USB-C cable to your PC's USB port
2. Install MobaXterm free Home Edition PC software from https://mobaxterm.mobatek.net/download.html
3. When the software runs, click **Start local terminal**
4. In the MobaXterm terminal window, type

```
nc xl3-usb.local 50312
```

5. Press ENTER when the password is requested
6. The response should be a message that identifies the XL3, for example:

```
XL3 Streaming API Text, A3A-00100-D0, 1.24
```

7. Enter the following command:

```
SOH
```

   The response could be

```
2;3;1698139974358;60000;13;LocalTime|TimeZone|BatterySOC|RunStatus|
WeatherStations|VDcIn|IPhantom|FreeStorage|GpsLocation|Temperature|AirPressure|
PowerSource|ClockSource;-|-|%|-|-|V|A|MB|deg|degC|hPa|-|-
3;3;1698139974398;2023-10-18 04:47:00|Europe/Berlin||Running|2|9.15|0.004|
1920.090||36.0|965.4|DcIn|NTP
```

   A new SOH (State of Health) data message appears every 60 seconds.
   Press CTRL+C to close the connection.

How to connect the XL3 physically

The XL3 can communicate through the following interfaces:

Interface	XL3 Address	Comment
USB-C	IP-Address or xl3-usb.local	One XL3 per PC
WiFi / Ethernet	IP-Address or xl3-00100.local	Ethernet using USB to Ethernet Adapter

Worldwide Access - NTi Connect

The NTi Connect service https://connect.nti-audio.com provides worldwide and secure access to the webpage, data files and APIs of XL3s.

XL3 uses port 22 to communicate with the NTi Connect server

NTi Connect provides free usage for data volumes up to 2 GB per month. For data consumption beyond this threshold, the download speed will be reduced. Opting for the "NTi Connect Open Data 365" subscription ensures uninterrupted communication at full speed. For comprehensive information, kindly consult the XL3 User Manual.

Communication Protocols

The APIs are available using the following communication protocols:

TCP

The APIs are available in a LAN over the following ports:

Service	TCP Port
Streaming API Advanced	50312, 50313
Streaming API Basic	not available
Control API	50300

Example

```
nc XL3-00228.local 50312
Password:
1234
XL3 Streaming API Text, A3A-00100-D0, 1.28
```

WebSockets

WebSockets use port 80/443 and are designed for communicating in WANs but are also available in LANs:

Service	WebSocket in a LAN
Streaming API	ws://*xl3_ip_address*/api/stream1/ ws://*xl3_ip_address*/api/stream2/ ws://*xl3_ip_address*/api/live/
Control API	ws://*xl3_ip_address*/api/control/

NTi Connect (see page 7) offers access to the API WebSockets from anywhere whenever an XL3 has access to the internet:

Service	WebSocket on NTi Connect
Streaming API	wss://connect.nti-audio.com/*ConnectKey*/api/stream1/ wss://connect.nti-audio.com/*ConnectKey*/api/stream2/ wss://connect.nti-audio.com/*ConnectKey*/api/live/
Control API	wss://connect.nti-audio.com/*ConnectKey*/api/control/

Communication - First steps

First Prompt

After a connection is established, the XL3 sends a first prompt, which could be:

Prompt	Action	Comment
`Password:`	Enter password. If the PW is wrong, the XL3 sends the message `Incorrect password` and closes the connection.	For a direct USB-C connection, any password will be accepted.
`Already in use`	The connection is closed. Retry to connect after some delay.	
`Busy, retry in a few seconds`		

Identification Message (Advanced Streaming)

After entering the correct password, the XL3 confirms with the interface identification message:

```
nc XL3-00228.local 50312
Password:
1234
NTi Audio XL3 Streaming API Text, A3A-00100-D0, 1.28
```

The interface is now ready for commands, for example:

```
SOH
2;3;1699971038797;60000;13;LocalTime|TimeZone|BatterySOC|RunStatus|WeatherStations|
VDcIn|IPhantom|FreeStorage|GpsLocation|Temperature|AirPressure|PowerSource|
ClockSource;-|-|%|-|-|V|A|MB|deg|degC|hPa|-|-
3;3;1699971000157;2023-11-14 15:10:00|Europe/Berlin|100.0|Stopped|0|9.10|0.006|
29792.281||33.7|967.2|DcIn|NTP
3;3;1699971060157;2023-11-14 15:11:00|Europe/Berlin|100.0|Running|0|9.11|0.006|
29769.656||33.7|967.1|DcIn|NTP
3;3;1699971120157;2023-11-14 15:12:00|Europe/Berlin|100.0|Running|0|9.11|0.006|
29747.000||33.8|967.1|DcIn|NTP
```

Basic Code examples (Python 3.8)

TCP

```
import socket

MyXL3 = "xl3-00100.local"
# MyXL3 = "192.168.201.100"
# MyXL3 = "xl3-usb.local"

XL3Password = b"1234"

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect((MyXL3, 50312))

# use makefile to provide a readline to the socket
sockFile = sock.makefile(mode='r')

passwordPrompt = sockFile.readline()
sock.send(XL3Password+b'\n')

print(sockFile.readline())      # Identification Message
sock.send(b"SOH\n")

while True:
    print(sockFile.readline())  # SOH header & results
```

WebSocket

```
from websocket import create_connection
# websocket --> install websocket-client
# https://github.com/websocket-client/websocket-client

MyXL3 = "xl3-00100.local"
# MyXL3 = "192.168.201.100"
# MyXL3 = "xl3-usb.local"

XL3Password = b"1234"

url = "ws://" + MyXL3 + "/api/stream1/"
xl3 = create_connection(url)

passwordPrompt = xl3.recv()
xl3.send(XL3Password+b'\n')

print(xl3.recv())       # Identification Message
xl3.send(b"SOH\n")

while True:
    print(xl3.recv())   # SOH header & results
```

NTi Connect

```
from websocket import create_connection
# websocket --> install websocket-client
# https://github.com/websocket-client/websocket-client

ConnectKey = "ABCDE-FGHIJ"
XL3Password = b"1234"

url = "wss://connect.nti-audio.com/" + ConnectKey + "/api/stream1/"
xl3 = create_connection(url)

passwordPrompt = xl3.recv()
xl3.send((XL3Password+b'\n'))

print(xl3.recv())       # Identification Message
xl3.send(b"SOH\n")

while True:
    print(xl3.recv())   # SOH header & results
```

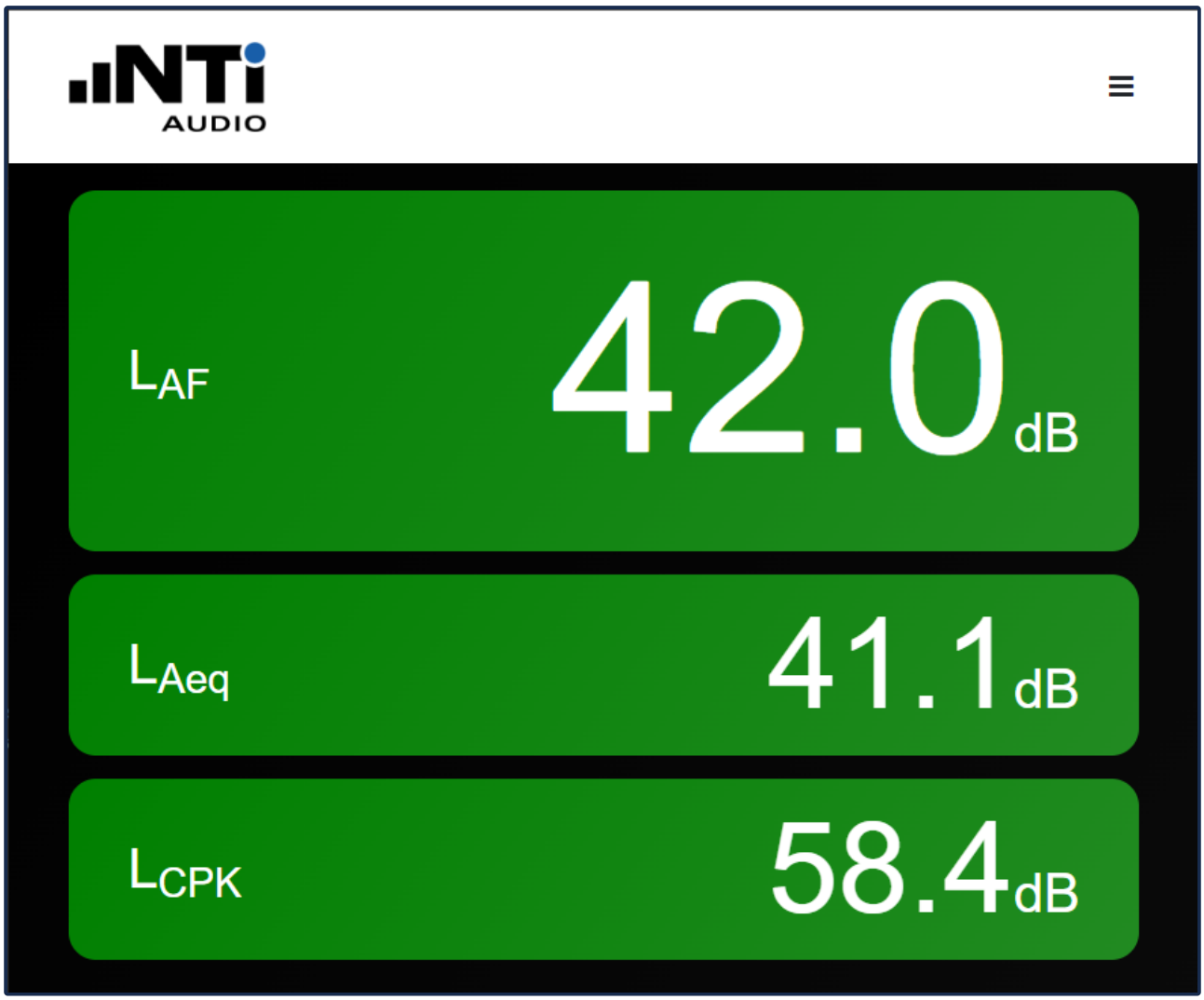
SOH means “State of Health” and returns a result every 60 seconds. A detailed description follows later in this document.

Streaming API

Basic Streaming API

Introduction

In the Sound Level Meter function, the XL3 can push the results of the 123 and spectrum screen in JSON format to up to 20 clients. This API is used for the XLVIEW of the XL3 webpage but is not limited to that use case.



Communication

After opening the connection to the API, the first prompt will either ask for the password or return `Already in use` if already 20 clients are connected (see First Prompt (see page 7)). After the first prompt, a command `START` must be sent. After that, the API sends

- a `settings` message. This settings is also sent whenever a setting on the XL3 is changed
- every 500ms a `data` message until the connection is closed.

Command

`START "options"`

Parameter	Parameter Type	Comment

`'options'`	STR	`123`
		Request the settings and results of 123 screen
		`123 Spectrum`
		Same as 123, but adds also the Spectrum

Answer

`JSON settings`
`JSON data` every 500ms

Example: 123 values

`START 123`

Settings:


```
{
  "error": "NONE",
  "settings": {
    "123": [{"name": "L`AF", "limit": {"on": false, "red": 100,"amber": 90}},
            {"name": "L`Aeq","limit": {"on": false, "red": 100,"amber": 90}},
            {"name": "L`AF", "limit": {"on": false, "red": 100,"amber": 90}},
            {"name": "L`CPK","limit": {"on": false, "red": 100,"amber": 90}},
            {"name": "L`AFmax","limit": {"on": false, "red": 100,"amber": 90}}],
    "device_name": "My XL3",
    "running": true,
    "timer": {"mode": "CONTINUOUS", "set": "00:00:00"}
  }
}
```


Data (every 500ms):


```
{
  "data": {
    "timer": "00:02:39",
    "123": [34.1, 43.0, 55.7, 31.6, 61.2]
  }
}
```


Values, that are not defined in the XL3 (“---”) are set to “null” in the JSON data

Example: 123 and spectrum values

`START 123 Spectrum`

Settings:


```
{
  "error": "NONE",
  "settings": {
    "123": [{"name": "L`AF", "limit": {"on": false, "red": 100,"amber": 90}},
            {"name": "L`Aeq","limit": {"on": false, "red": 100,"amber": 90}},
            {"name": "L`AF", "limit": {"on": false, "red": 100,"amber": 90}},
            {"name": "L`CPK","limit": {"on": false, "red": 100,"amber": 90}},
            {"name": "L`AFmax","limit": {"on": false, "red": 100,"amber": 90}}],
    "device_name": "My XL3",
    "running": true,
    "timer": {"mode": "CONTINUOUS", "set": "00:00:00"},
    "spectrum": {"name": "live","octres": "1/3", "bands": 36}
  }
}
```


Data (every 500ms):


```
{
  "data": {
    "timer": "00:02:39",
    "123": [34.1, 43.0, 55.7, 31.6, 61.2],
    "spectrum": [34.3, 45.6, 52.8, 49.0, 46.0, 38.2, 35.0, 31.3, 30.0, 33.5,
                 28.2, 40.9, 40.6, 38.7, 40.1, 39.6, 27.7, 27.3, 19.2, 18.8,
                 22.5, 18.1, 18.7, 20.3, 16.9, 17.9, 14.5, 19.4, 19.2, 17.4,
                 16.8, 15.1, 15.0, 12.4, 10.0, 14.2]
  }
}
```

Message	Parameter	
settings	error	`NONE` no error `NOT_IN_SLM` Basic streaming is only supported in the Sound Level Meter function
	name	Indicator name containing supscript info for formatting. For example `L`AF´+k1` should be printed like $L_{AF}$ +k1 Typical code in html for printing the name: `name.replace(/`/g, '<sub>').replace(/´/g, '</sub>')`
	running	Measurement is running, `true` or `false`
	timer / mode	Setting of the measurement timer `CONTINUOUS`, `SINGLE`, `REPEATED_SYNC`
	timer / set	Duration setup for the timer
data	timer	Current value of the measurement timer

Python example

```
from websocket import create_connection
# websocket --> install websocket-client (https://github.com/websocket-client/
websocket-client)

ConnectKey = "ABCDE-FGHIJ"
XL3Password = b"1234"

url = "wss://connect.nti-audio.com/" + ConnectKey + "/api/live/"
xl3 = create_connection(url)

passwordPrompt = xl3.recv()
xl3.send((XL3Password+b'\n'))

xl3.send(b"START 123\n")

while True:
    print(xl3.recv())       # Settings & results
```

Advanced Streaming API

Introduction

The XL3 can stream SPL results, spectral data, SOH (state of health) and the data of a connected weather station to a client, typically a server running a Noise Monitoring service. Besides live data, the XL3 also supports retrieving historical data. So, when a server loses the connection to an XL3, no relevant data is lost and can be requested when the connection is established again.

The Streaming API requires some settings on the XL3, which can not be set via the UI. They are set by loading a configuration file.

Configuring the XL3

Configurations of the XL3 are organized in JSON format. When the XL3 loads a configuration, it first sets all parameters to default and then loads the JSON.

The internal storage of the XL3 contains two helpful files:

File	Comment
`Configurations/default.xl3cfg`	Default values for all XL3 settings
`Configurations/documentation.txt`	Available values for all XL3 settings

Typical configuration file

The content of a typical configuration file when using the streaming API:

```
{ "MeasurementID": "SLM", "UUID": "00000000-0000-0000-0000-000000000000" }
##CONFIG:
{
    "system_settings": {
        "connections": {
            "watchdog": "on"
        },
        "measurement": {
            "save_mode": "automatic"
        }
    },
    "SLM": {
        "auto_run": "on",
        "report_logging": {
            "logging": {
                "utc_for_text_files": "off",
                "file_format": "binary",
                "folder_struct": "noisemonitoring",
                "audio": {
                    "mode": "on"
                }
            },
            "spectra": "eq, max, min"
        }
    }
}
```

Important Settings for Streaming

Setting	Description
`watchdog`	`on` The XL3 will automatically restart if there is no internet connection for one hour. A maximum of two restarts is executed within 12 hours. An eye symbol in the top bar of the XL3 display represents the activation of the watchdog. When a network connection is active, the eye symbol toggles with the network symbol. ℹ We recommend switching this function on for unattended noise monitoring applications.
	`off` (default) Watchdog is off

`save_mode`	`automatic`
	Measurement is saved automatically. In this mode, the file push service can push each completed file.
	`assisted` (default) `manually`
	Not recommended for streaming or unattended noise monitoring.

`auto_run`
 	
If the Measurement scheduler (see page 7) is active, this setting has no effect.
	`on`
	• the XL3 will automatically start the measurement when the configuration is activated.
	• in case of an error (e.g. if the storage is unavailable), the XL3 tries to restart the measurement every 10 seconds.
	• the measurement can be stopped manually, for example, to execute a calibration. During this "*local operation*"
	  • the measurement is automatically started again if there was no user interaction for 60 seconds (as long the configurations was not exited by the operator).
	  • Configuration sent via the the Streaming API are rejected. See Local operation (see page 30) for more details.

	`off` (default) If auto_run was on before, an ongoing measurement will be stopped and saved when the configuration is received.
`file_format`	`text_tsv` (default) Tabulator separated text files (default)

Setting	Description
	`binary` Required for streaming SPLLOG. XL3 can only stream data from binary files. Binary files contain the same information as text files but can be accessed faster by machines.
	`text_tsv + binary` Both file types are generated simultaneously
`folder_structure`	`standard` (default) Measurements are organized in project folders (default)
	`noisemonitoring` Mandatory for streaming: • Measurements are organized in YEAR/MONTH, allowing machines to find the correct files quickly. • Data is stored based on UTC. • When the free space on the storage gets below 1GByte, the oldest measurement folder and its content are deleted.
`utc_for_text_files`	Specifies which time zone is used for • text files • the Broadcast Audio Extension Chunk of wave files. `off` (default) Local time `on` UTC

Measurement Scheduler

Description

The XL3 can start and stop measurements daily at a specific time and optionally power off during times when it is not measuring. This mode is only accessible via the configuration file.

```
{
  "MeasurementID": "SLM",
  "UUID": "00000000-0000-0000-0000-000000000000"
}
##CONFIG:
{
  "SLM": {
    "scheduler": {
      "select": "daily",
      "daily": {
        "start_time": {
          "hour": 8,
          "minute": 15
        },
        "end_time": {
          "hour": 11,
          "minute": 30
        }
      },
      "power_save": "off",
      "power_save_delay_minutes": 0
    }
  }
}
```

	Setting	Description
1	`select`	`off` (default) Scheduler is disabled
2		`daily` Scheduler starts and stops the measurement daily at the specified time.

	Setting	Description

3	`start_time`
	`end_time`
		Daily start/end time of the measurement
		`hour`
		Integer value from 0 to 23
		`minute`
		Integer value from 0 to 59
		
		If the start time is set later than the end time, the measurement continues past midnight into the next day.

4	`power_save`	`off` (default) The XL3 remains powered on after the measurement ends
5		`on` The XL3 powers off after the measurement ends and powers back on shortly before the next measurement starts. This saves energy between measurements. If less than ten minutes remain before the next measurement, the XL3 will not power off.
6	`power_save_delay_minutes`	Integer value from 0 to 1428. The shutdown of the XL3 after the measurement is delayed by this time. This allows measurement data to be downloaded from the device before it shuts down. If the configuration is set so that less than ten minutes remain between the planned shutdown and the next measurement, the XL3 will not power off.

Manual Operation

- When manual operation is active with an enabled scheduler, no configuration can be loaded via the streaming or control API. Instead, error 1050 (local operation) is returned.
- Measurements cannot be started outside scheduled times.
- A running measurement can be interrupted for calibration. It can be restarted manually or will resume automatically after one minute of inactivity.
- Calibration is possible during the shutdown delay after a measurement. If the delay has expired, local operation can extend it, preventing the XL3 from shutting down during calibration.

Power Management

- If manually powered on during a power-saving phase, the XL3 will automatically power off after one minute of inactivity, unless the next measurement is less than 10 minutes away.
- Manual power-off is blocked when the scheduler is active.
- Emergency shutdown (long press of the power button) remains possible.
  - WARNING: This disables automatic restart for the next measurement.

User Interface

Symbol	Description
	Measurement Scheduler Symbol
	Measurement is not running, but the measurement scheduler controls XL3.
	Logging while the scheduler is active.
	Status bar: Measurement scheduler is active. Replaces the “normal” logging symbol.

Data Stream Channels

Channels ID

The streaming API is organized through the following channels:

Channel	Channel ID	Description	Live Data	Historical Data
SYSTEM	0	Commands and errors		
SPLLOG	1	SLM log data	Error	Error
SPLREP	5	SLM report data	Error	Error
AUDIO	2	Audio data		Error
SOH	3	State of health data	Error	
WEATHER	4	Weather data	Error	Error

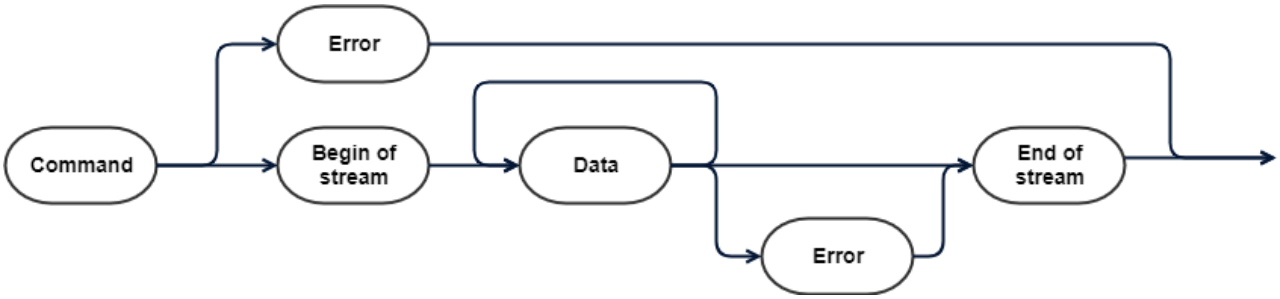
Content ID

Channels need to be opened once by a command. The device then sends responses of the following type:

Content-Type	Content ID	Description
Error	1	Error message with error binary error number and ASCII error text
Begin of stream	2	The package contains begin of stream data, usually the data header.
Data	3	Channel-dependent streamed data
End of stream	4	Marks the end of a stream

Channel states

Channels follow this flow:



Terminate commands with new line “\n”

XL3 Message format

Messages sent by the XL3 follow this format:

Contend ID	;	Channel ID	;	Time Stamp	;	...

Messages from XL3 are designed to be first split by `;`.
Then, split individual packets by `|`

SYSTEM Channel

Command:

`SYST:CONF 'Name', Base64ConfigData`

Configures the XL3 with the Base64ConfigData.

Parameter	Parameter Type	Comment
`'Name'`	QSTR	Name of the file used to store the configuration in the Configuration folder of the XL3. Encode CP 1252.
`Base64ConfigData`	STR	Base64 coded configuration files. For example, Typical configuration file (see page 21) coded as Base64.

Answer:

`1;0;ERROR_NUMBER;ERROR_TEXT`

Example:

```
SYST:CONF
'MyConfig',eyAiTWVhc3VyZW1lbnRJRCI6ICJTTE0iLCAiVVVJRCI6ICIwMDAwMDAwMC0wMDAwLTAwMDAtMDA
wMC0wMDAwMDAwMDAwMDAiIH0NCiMjQ09ORklHOg0Kew0KICAgICJzeXN0ZW1fc2V0dGluZ3MiOiB7DQogICAgI
CAgICJjb25uZWN0aW9ucyI6IHsNCiAgICAgICAgICAgICJ3YXRjaGRvZyI6ICJvbiINCiAgICAgICAgfSwNCiA
gICAgICAgIm1lYXN1cmVtZW50Ijogew0KICAgICAgICAgICAgInNhdmVfbW9kZSI6ICJhdXRvbWF0aWMiDQogI
CAgICAgIH0NCiAgICB9LA0KICAgICJTTE0iOiB7DQogICAgICAgICJhdXRvX3J1biI6ICJvbiIsDQogICAgICA
gICJyZXBvcnRfbG9nZ luZyI6IHsNCiAgICAgICAgICAgICJsb2dnaW5nIjogew0KICAgICAgICAgICAgICI
CJmaWxlX2Zvcm1hdCI6ICJiaW5hcnkiLA0KICAgICAgICAgICAgICJmb2xkZXJfc3RydWN0IjogIm5vaXN
lbW9uaXRvcmluZyIsDQogICAgICAgICAgICAgImF1ZGlvIjogew0KICAgICAgICAgICAgICAgICAgICAib
W9kZSI6ICJvbiINCiAgICAgICAgICAgICAgICB9DQogICAgICAgICAgICB9LA0KICAgICAgICAgICAgInNwZWN
0cmEiOiAiZXEsIG1heCwgbWluIg0KICAgICAgICB9DQogICAgfQ0KfQ0K
1;0;0;OK
```

Steps executed by this command:

- Decodes the `Base64ConfigData` and stores it in the XL3 Configurations folder using the name `Name`.xl3cfg.
- Checks if applying configuration requires restarting an ongoing measurement. If yes, stop the measurement.
- Load the configuration.
- Restart the measurement if it was stopped before (or if `auto_run` is set to `on` in the configuration).

It is good practice to send this command whenever the Streaming API WebSocket to the device is opened.

Python example:

```
import base64
from websocket import create_connection
# websocket --> install websocket-client (https://github.com/websocket-client/
websocket-client)

def getConfigFileAsBase64(file_name):
    """Reads a file and returns its content as base64 encoded string"""
    with open(file_name, 'rb') as file:
        file_content = file.read()
        base64_content_s = base64.b64encode(file_content)
        return base64_content_s

ConnectKey = "ABCDE-FGHIJ"
XL3Password = b"1234"

url = "wss://connect.nti-audio.com/" + ConnectKey + "/api/stream1/"
xl3 = create_connection(url)

passwordPrompt = xl3.recv()
xl3.send((XL3Password+b'\n'))

print(xl3.recv())           # Identification Message

ConfigBase64 = getConfigFileAsBase64("MyConfigFile.xl3cfg")
q = b"SYST:CONF 'ApiDemo'," + ConfigBase64 + b"\n"
xl3.send(q)
print(xl3.recv())
```

Local operation when a configuration is active

When `auto_run` is configured, and the operator locally stops the measurement (for example, to execute a calibration), sending SYST:CONF will result in an error 1050 (Loading configuration failed: Local operation active). The XL3 will accept the command again

- once the measurement was restarted, either manually by the operator or automatically through auto_run
- the operator exited the configuration (which will disable auto_run)

SPLLOG Channel

Command:

`SPLLOG START_TIME_REQ, "Indicators List" [, MAX_HISTORY_LINES]`

Request results from the SLM logging data starting from the START_TIME_REQ time.

Parameter	Parameter Type	Comment
`START_TIME_REQ`	INT	UNIX timestamp * 1000 (ms) https://www.epochconverter.com/
`'Indicators List'`	QSTR	List of values required as a quoted string e.g. “LAEQ LAFMAX“
`MAX_HISTORY_LINES`	INT	**Optional Parameter**: Specifies the maximum number of lines of history data that will be streamed before the stream will stop, valid range is 10 to 1000, or -1 if as many data lines as possible should be streamed (if parameter not given: default is 1000). Note: If value is out of valid range the value is automatically and silently adjusted. Note: This limitation takes only effect for history data and not for live data.

Answer:

`2;1;START_TIME_CONF;INTERVAL;NUMBER_OF_INDICATORS;INDICATOR_NAME|INDICATOR_NAME| ...`

`3;1;TIMESTAMP;RESULT|RESULT ...`

A gap in data:

The XL3 sends data in fixed `INTERVAL` s starting from `START_TIME_CONF`. The `TIMESTAMP` is always one `INTERVAL` bigger than the `TIMESTAMP` sent before.

If there is a gap in the data (e.g. because the measurement was stopped and restarted), the channel will close with an “End of stream” message. When this happens, send the command again with the last received `TIMESTAMP` as `START_TIME_REQ`. The XL3 will then start delivering data after the end of the gap.

Example:

```
SPLLOG 1690196106000, "LAEQ LAFMAX"\n
2;1;1690196106000;1000;2;LAEQ|LAFMAX
3;1;1690196107000;45.0|51.4
3;1;1690196108000;34.8|38.3
...
```

```
3;1;16901961321000;34.8|38.3
4;1
SPLLOG 16901961321000, "LAEQ LAFMAX"\n
2;1;1690898194000;1000;2;LAEQ|LAFMAX
3;1;1690898195000;65.4|67.8
3;1;1690898196000;57.8|59.2
```

Example with optional parameter:

```
spllog 1710928800000, "laeq lafmax", 20\n
2;1;1710928800000;1000;2;LAEQ|LAFMAX
3;1;1710928801000;34.6|35.0
3;1;1710928802000;34.7|35.3
3;1;1710928803000;35.1|35.9
...
3;1;1710928820000;34.6|36.5
4;1
```

Examples of failed requests:

A request with an unknown indicator “ABC”:

```
SPLLOG 1690288491000, "ABC"\n
1;1;40;Wrong type of parameter(s)
```

Requesting data from “now”, but no measurement is running:

```
SPLLOG 1690288809000, "LAEQ LAFMAX"\n
1;1;10000;NO DATA FOUND ERROR 1
```

Indicator List:

Group	Available indicators
Broadband	`LxyMAX` `LxyMIN` `LxIMAX` `LxIMIN` `LxFINST` `LxEQ` `LxIEQ` `LxE` `LxPKMAX` `LAFT3` `LAFT5` `LAEQ_G1` `LAEQ_G2` `LAEQ_G3` `LAEQ_G4` `LCEQ_G1` `LCEQ_G2` `LCEQ_G3` `LCEQ_G4` `LN1` `...` `LN7` `LCEQ_G1-LAEQ_G1` `LCEQ_G2-LAEQ_G2` `LCEQ_G3-LAEQ_G3` `LCEQ_G4-LAEQ_G4`

Group	Available indicators

Spectrum
All bands
	1/1 octave resolution:
	`OCT_xEQ` `OCT_xyMIN` `OCT_xyMAX`
	`OCT_xyN1` `...` `OCT_xyN7`
	`OCT_xyINST`
	
	1/3 octave resolution:
	`3OCT_xEQ` `3OCT_xyMIN` `3OCT_xyMAX`
	`3OCT_xyN1` `...` `3OCT_xyN7`
	`3OCT_xyINST`

Spectrum
Individual bands
	1/1 octave resolution, # = 1 … 12 (8Hz … 16kHz):
	`OCT#_xEQ` `OCT#_xyMIN` `OCT#_xyMAX`
	`OCT#_xyN1` `...` `OCT#_xyN7`
	`OCT#_xyINST`
	
	1/3 octave resolution, # = 1 … 36 (6.3Hz … 20kHz):
	`3OCT#_xEQ` `3OCT#_xyMIN` `3OCT#_xyMAX`
	`3OCT#_xyN1` `...` `3OCT#_xyN7`
	`3OCT#_xyINST`

Noise Locator	Values representing results:
	LP … Sound Pressure Level A-Weighted
	LI … Sound Intensity Level A-Weighted
	DPI … Delta PI (difference of LP - LI)
	AZ … Azimuth
	EL … Elevation
	Order of values if the indicator returns a four value tupel, values are separated by a pipe:
	|LP|DPI|AZ|EL| , so for example |78.3|10.9|156|10|
	Broadband (four value tupel):
	`NL`
	Individual broadband values (single value):
	`NL_LP NL_LI NL_DPI NL_AZ NL_EL`
	1/1 octave all bands (7 bands, 7 four value tupels)
	`OCT_NL`
	1/1 octave, Individual bands, # = 1 ... 7 (63Hz … 4kHz, four value tupel)
	`OCT#_NL`
	Individual values of individual bands (single value)
	`OCT#_NL_LP OCT#_NL_LI OCT#_NL_DPI OCT#_NL_AZ OCT#_NL_EL`

x = [A|C|Z]
y = [F|S]

278 results can be streamed. A spectrum counts as 36 for 1/3 octave resolution and 12 for 1/1 octave resolution.

```
spllog 1769700511000, "LAEQ NL"
2;1;1769700514000;2026-01-29;15:28:34 000000;100;5;LAEQ|NL
3;1;1769700514100;2026-01-29;15:28:34 100000;17.4|67.5|3.5|71|3
```

With the default settings,

- spectral data is available in 1/3 octave resolution, Z frequency weighting and F time weighting.
- Gliding LEQs (e.g. LAEQ_G1) have 5sec, 10min, 15min and 60min averaging time.
- Level Statistics (e.g. LN1) are set to 1.0%, 5.0%, 10.0%, 50.0%, 90.0%, 95.0% and 99.0%.
  Broadband statistics are based on LAF, spectral statistics on the frequency and time weighting of the spectrum (“fwtw”)

To change these settings, adapt and use this configuration:

```
{  "MeasurementID": "SLM", "UUID": "00000000-0000-0000-0000-000000000000" }
##CONFIG:
{
    "SLM": {
        "auto_run": "on",
        "report_logging": {
            "logging": {
                "utc_for_text_files": "off",
                "file_format": "binary",
                "folder_struct": "noisemonitoring",
                "audio": {
                    "mode": "on"
                }
            },
            "spectra": "eq, max, min"
        },
        "spectrum": {
            "octres": "1/3",
            "hold_time": "3 sec",
            "fwtw": "ZF"
        },
        "gliding_eq": {
            "average_time": {
                "1": "5 sec",
                "2": "10 min",
                "3": "15 min",
                "4": "60 min"
            }
        },
        "percentile": {
            "1": 1.0,
            "2": 5.0,
            "3": 10.0,
            "4": 50.0,
            "5": 90.0,
            "6": 95.0,
            "7": 99.0,
            "source": "AF"
        }
    }
}
```

SPLREP Channel

Command:

```
SPLREP START_TIME_REQ, "Indicators List" [, MAX_HISTORY_LINES]
```

Request results from the report data starting from the START_TIME_REQ time.

The XL3 records aggregated report data when the timer mode is set to “Repeated”.

```
{  "MeasurementID": "SLM", "UUID": "00000000-0000-0000-0000-000000000000" }
##CONFIG:
{
    "SLM": {
        "timer": {
            "mode": "Repeated",
            "set": {
                "repeated_ms": 300000
            }
        }
    }
}
```

Parameter	Parameter Type	Comment
`START_TIME_REQ`	INT	UNIX timestamp * 1000 (ms) https://www.epochconverter.com/
`"Indicators List"`	QSTR	List of values required as a quoted string e.g. “LAEQ LAFMAX“ Indicator list (see page 32)
`MAX_HISTORY_LINES`	INT	**Optional Parameter**: Specifies the maximum number of lines of history data that will be streamed before the stream will stop, valid range is 10 to 1000, or -1 if as many data lines as possible should be streamed (if parameter not given: default is 1000). Note: If value is out of valid range the value is automatically and silently adjusted. Note: This limitation takes only effect for history data and not for live data.

Answer:

```
2;5;START_TIME_CONF;0;NUMBER_OF_INDICATORS;INDICATOR_NAME|INDICATOR_NAME| ...
```

```
3;5;TIMESTAMP;DURATION;RESULT|RESULT ...
```

When a measurement is started and/or stopped, the `DURATION` of the first and the last data message may differ from the set `repeated_ms` duration.

A gap in data:

The XL3 sends data starting from `START_TIME_CONF`. After the first data line, the `TIMESTAMP` is always one `DURATION` bigger than the `TIMESTAMP` sent before.

If there is a gap in the data (e.g. because the measurement was stopped and restarted), the channel will close with an “End of stream” message. When this happens, send the command again with the last received `TIMESTAMP` + `DURATION` as `START_TIME_REQ`. The XL3 will then start delivering data after the end of the gap.

⚠ Continuing after a gap is different between SPLLOG and SPLREP

Example:

```
SPLREP 1695214350000, "LAEQ LAFMAX"\n
2;5;1695214372000;0;2;LAEQ|LAFMAX
3;5;1695214372000:8000;45.0|51.4
3;5;1695214380000;15000;34.8|38.3
3;5;1695214395000;15000;51.1|69.8
...
3;5;1695214770000;15000;48.8|60.8
3;5;1695214785000;11000;34.8|38.3
4;5
SPLREP 1695214796000, "LAEQ LAFMAX"\n
2;5;1695214800000;0;2;LAEQ|LAFMAX
3;5;1695214800000;15000;65.4|67.8
3;5;1695214815000;15000;57.8|59.2
```

AUDIO Channel

Command:

`AUDIO START_TIME_REQ, DURATION`

Request audio data from `START_TIME_REQ` with the length `DURATION` (in seconds)

Answer:

```
2;2;BASE64_WAVE_FILE_HEADER
3;2;BASE64_WAVE_FILE_DATA
...
3;2;BASE64_WAVE_FILE_DATA
4;2
```

To get an actual wave file, decode the `BASE64_WAVE_FILE_HEADER` and the `BASE64_WAVE_FILE_DATA` and store the result in a file with the ending ".wav".

The wave file includes a Broadcast Audio Extension Chunk, which describes the content of the wave file. See https://tech.ebu.ch/docs/tech/tech3285.pdf for a full specification. The chunk also includes date and time information, which is either in local time or UTC, dependent on the "utc_for_text_files" setting.

Example:

```
Audio 1690547362000, 3\n
2;2;UklGRviRAABXQVZFZm10IBQAAAARAAEAwF0AAIYvAAAAAQQAAgD5AWZhY3QEAAA...
3;2;7Oc2AKSMQpGZGCOq0g4CCIqSWcFSE8mLAggouEMRiLAsR5iJQso1MMHKUaKdMaO...
3;2;mAohALqSrFCZ6QEIi5mZ8w0ImZwDyI5AwQExkJ2wqZ0ZA9uaQZkwFwgqgZgh2Dj...
...
3;2;B94nAEgYggBDUSE0BBkiQQFiIRMhMiMMJyIIJDGBJWCEOJoCs0lZkiUJQCGUQkk...
4;2
```

Getting Audio - Programming Example (Python 3.8):

```
from websocket import create_connection
import time
import base64

ConnectKey = "ABCDE-FGHIJ"
url = "wss://connect.nti-audio.com/api/" + ConnectKey + "/stream2/"

ws = create_connection(url)

passwordPrompt = ws.recv()
ws.send(b'1234\n')
identificationMessage = ws.recv()
print(identificationMessage)

now = (int(time.time()) - 60) * 1000
q = b'Audio %d, 3\n' % now
ws.send(q)
print(q)

response = ws.recv().decode("utf-8").strip()
print(response)

if response[0] == "2":  # Begin of stream
    waveFile = base64.b64decode(response[4:])

    while True:
        response = ws.recv().decode("utf-8").strip()

        if response[0] == "3":      # Data
            waveFile += base64.b64decode(response[4:])

        if response[0] == "4":      # End of stream
            with open("xl3-demo.wav", "wb") as file:
                file.write(waveFile)
            break

ws.close()
```

SOH Channel

Command:

`SOH`

Request streaming of SOH data.

Answer:

`2;3;START_TIME_CONF;INTERVAL;NUMBER_OF_RESULTS;RESULT_LIST;RESULT_UNITS`

Parameter	Example	Unit	Comment
`INTERVAL`	60000	ms	Always 60000
`NUMBER_OF_RESULTS`	13		Always 13
`LocalTime`	2023-07-24 12:55:00		
`TimeZone`	Europe/Berlin		
`BatterySOC`	100	%	State of charge of the battery pack (0 .. 100) Empty if there is no battery inserted
`RunStatus`	RUNNING		
`WeatherStations`	0		Number of detected weather stations (0..3)
`VDcIn`	9.67	V	Voltage on the power supply connector
`IPhantom`	0.011	A	Current consumption of the connected microphone.
`FreeStorage`	283397.625	MB	
`GpsLocation`	47.176090 9.512550	Degree	Latitude Longitude (requires optional external GPS receiver)
`Temperature`	36.0	°C	Temperature inside of the XL3
`AirPressure`	965.4	hPa	

Parameter	Example	Unit	Comment
`PowerSource`	DcIn		External power source connected to XL3: `None`, `USB` or `DcIn`
`ClockSource`	NTP		Source of the internal clock `Internal`, `NTP`, `GPS`, `PPS`
`CpuTemperature`	59.2	°C	CPU Temperature
`CicTime`	2025-11-06 15:29:05		Date/Time of the last CIC measurement or verification
`CicStatus`	Passed		Status of the last CIC measurement or verification: `Done`, `Passed`, `Failed`, `NoReference`, `NoVerification`, `NoAsd`, `TooNoisy`, `NotSupported`, `InternalError`
`CicDeltaF1`	-0.00	dB	Level deviation to last reference measuremen for f1
`CicDeltaF2`	0.00	dB	Level deviation to last reference measuremen for f2
`NoiseLocator`	1		1 if a Noise Locator is connected otherwise 0

When fields of the result cannot be determined, they will be left blank.

Example:

```
SOH\n
2;3;1698139974358;60000;13;LocalTime|TimeZone|BatterySOC|RunStatus|WeatherStations|
VDcIn|IPhantom|FreeStorage|GpsLocation|Temperature|AirPressure|PowerSource|
ClockSource|CpuTemperature|CicTime|CicStatus|CicDeltaF1|CicDeltaF2|NoiseLocator;-|-|
%|-|-|V|A|MB|deg|degC|hPa|-|-|degC|-|-|dB|dB|-
3;3;1698139974398;2023-10-18 04:47:00|Europe/Berlin|100.0|Running|2|9.15|0.004|
1920.090|47.176090 9.512550|36.0|965.4|DcIn|NTP|59.7|2025-11-06 15:29:05|Passed|-0.00|
0.00|1
```

WEATHER Channel

Command:

```
WEATHER START_TIME_REQ [, MAX_HISTORY_LINES]
```

Request results from the Weather logging data starting from the START_TIME_REQ time.

Parameter	Parameter Type	Comment
`START_TIME_REQ`	INT	UNIX timestamp * 1000 (ms) https://www.epochconverter.com/
`MAX_HISTORY_LINES`	INT	**Optional Parameter**: Specifies the maximum number of lines of history data that will be streamed before the stream will stop, valid range is 10 to 1000, or -1 if as many data lines as possible should be streamed (if parameter not given: default is 1000). Note: If value is out of valid range the value is automatically and silently adjusted. Note: This limitation takes only effect for history data and not for live data.

Answer:

```
2;4;START_TIME_CONF;INTERVAL;NUMBER_OF_WEATHER_STATIONS;...
```

```
3;4;TIMESTAMP;RESULT|RESULT ...
```

If there is a gap in the recorded data, the behaviour is similar to that described for the SPLLOG channel (see page 31).

Example:

```
WEATHER 1701429720000
2;4;1701426240000;60000;1;Vaisala;USB;R3750165;6;S1|S1|S1|S1|S1|S1;Speed_Min|
Speed_Avg|Speed_Max|Dir_Min|Dir_Avg|Dir_Max;m/s|m/s|m/s|deg|deg|deg
3;4;1701426300000;0.00|0.00|0.10|90.00|86.00|180.00
3;4;1701426360000;0.00|0.00|0.10|90.00|91.00|330.00
```

When two weather stations are connected:

```
WEATHER 1701429720000
2;4;1701426240000;60000;2;Vaisala|LCJ Capteur;USB|SDI-12.A1;R3750165|00000;12;S1|S1|
S1|S1|S1|S1|S2|S2|S2|S2|S2|S2;Speed_Min|Speed_Avg|Speed_Max|Dir_Min|Dir_Avg|Dir_Max|
Speed_Min|Speed_Avg|Speed_Max|Dir_Min|Dir_Avg|Dir_Max;m/s|m/s|m/s|deg|deg|deg|m/s|m/s|
m/s|deg|deg|deg'
```

```
3;4;1701426300000;0.00|0.00|0.10|90.00|86.00|180.00|0.20|0.40|4.50|188.00|155.00|
133.00
3;4;1701426360000;0.00|0.00|0.10|90.00|91.00|330.00|0.20|0.40|4.50|188.00|155.00|
133.00
```

Using multiple channels

… on the same WebSocket

Multiple channels can be used on the same WebSocket:

Example:

```
NTi Audio XL3 Streaming API Text, A3A-00100-D0, 1.28
SYST:CONF
'MyConfig',eyAgIk1lYXN1cmVtZW50SUQiOiAiU0xNIiwgIlVVSUQiOiAiMDAwMDAwMDAtMDAwMC0wMDAwLTA
wMDAtMDAwMDAwMDAwMDAwIiB9DQojI0NPTkZJRzoNCnsNCiAgICAiU0xNIjogew0KCQkiYXV0b19ydW4iOiAib
24iLA0KICAgICAgICAic3BlY3RydW0iOiB7DQogICAgICAgICAgICAib2N0cmVzIjogIjEvMyIsDQogICAgICA
gICAgICAiZnd0dyI6ICJaRiINCiAgICAgICAgfSwNCiAgICAgICAgInJlcG9ydF9sb2dnaW5nIjogew0KICAgI
CAgICAgICAgImxvZ2dpbmciOiB7DQogICAgICAgICAgICAgICAgImZpbGVfZm9ybWF0IjogImJpbmFyeSIsDQo
gICAgICAgICAgICAgICAgImZvbGRlcl9zdHJ1Y3QiOiAibm9pc2Vtb25pdG9yaW5nIiwNCiAgICAgICAgICAgI
CAgICAiYXVkaW8iOiB7DQogICAgICAgICAgICAgICAgICAgICJtb2RlIjogIm9uIg0KICAgICAgICAgICAgICA
gIH0NCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgI
CAgIH0NCiAgICAgICB9DQp9DQo=
SPLLOG 1690196106000, "LAEQ LAFMAX 3oct_zeq"
SOH
WEATHER 1690196106000
1;0;0;OK
2;1;1690196106000;1000;38;LAEQ|LAFMAX
2;3;1690196107036;60000;9;LocalTime|TimeZone|BatterySOC|RunStatus|WeatherStations|
VDcIn|IPhantom|FreeStorage|GpsLocation;-|-|%|-|-|V|A|MB|deg
3;3;1690196100244;2023-07-24 12:55:00|Europe/Berlin|100.0|Running|1|9.67|0.011|
483397.625|
2;4;1690196107000;60000;1;Vaisala;USB;R3750165;6;S1|S1|S1|S1|S1|S1;Speed_Min|
Speed_Avg|Speed_Max|Dir_Min|Dir_Avg|Dir_Max;m/s|m/s|m/s|deg|deg|deg
3;4;1690195107000;0.00|0.10|0.20|90.00|186.00|270.00
3;1;1690196107000;42.1|48.3
3;1;1690196108000;38.0|42.4
3;1;1690196109000;33.2|36.7
3;1;1690196110000;33.4|34.2
3;1;1690196111000;38.7|44.1
3;1;1690196112000;33.5|38.9
3;1;1690196113000;32.8|34.6
```

… on different WebSockets

The Streaming API offers two TCP ports / two WebSockets, which can be used simultaneously with the restriction that only one channel may be opened simultaneously.

Example:

- WebSocket1 could be used to stream SOH, SPLLOG and Weather data.
- WebSocket 2 could be used for querying Audio.
  Opening a SOH stream on WebSocket 2 would result in the following error message:
  `1;3;9001;Attempt to use an already opened channel: Stream SOH Channel\n'`

Control API

Introduction

The XL3 can be controlled and queried remotely using the Control API from external client software with a command set. The command set allows you to set up the device, Start/Stop measurements, and retrieve measurement results. This interface is typically used for measurement applications like acoustic test stands or end-of-line testing.

Command Structure

Remote commands are sent in ASCII format to the XL3. The line feed character (LF, 0x0A) is the message terminator for XL3 commands. So, every command transmission from your PC to your XL3 or vice versa has to be terminated with a line feed LF. The measurement commands are divided into subsystems (i.e. logical groups).

Subsystem	Function
*	Device status common commands
INITiate	Status control for a measurement
MEASurement	Measurement result query commands
INPut	Sensor settings command
CALIbrate	Sensor calibration commands
ASD	Microphone Automatic Sensor Detection commands
SYSTem	System status commands

Command Format

The XL3 accepts either total keywords (long form) of commands or their abbreviations (short form). In the command list, the CAPITAL letters indicate the abbreviation. However, the XL3 accepts lowercase and UPPERCASE letters as input, i.e. commands and parameters are not case-sensitive.

The command description contains special symbols (see page 93) and parameter types (see page 93).

Unquoted strings in commands may not contain blanks or other characters, which may be interpreted as control characters or separators. Strings embedded in quotation marks may include all characters except the message termination character (LF).

Command Responses

Commands that query the XL3 for values have a “?“ at the end of the command. Query commands return a response (Answer). Commands that set values in the XL3 have no “?“. Set commands do not return a response.

> XL3 responds to each command with LF. The XL3 sends the command response only after the command is executed by the XL3. In other words, until the LF is returned, no other command can be processed by the XL3.

This is especially helpful for synchronizing commands that need a little time. e.g. for the `INIT START` command, the answer LF is sent when the measurement starts (after the settling time reaches 0). This makes it easy to synchronize further actions.

When there is no response, something went wrong. Please check the error queue (see page 47).

Error Queue

Errors are stored in the error queue and can be queried with the SYSTem:ERRor? (see page 83) command. If an error occurs during the execution of a command or a query, a corresponding error is pushed into the error queue. A semicolon is returned instead of the expected answer if the command is a query.

Multiple Commands

Multiple commands separated by semicolons ";" are supported, e.g. `meas:init;:syst:err?`

Notice that a command tree reset character (“:” as shown in the example above) is required for a subsequent command if the last node is not re-used. The commands are processed one after the other. The following command is only executed when the last one is completed. The response of multiple commands is separated by “;” and one LF at the line end. If there is a semicolon without a preceding value, it is an error. Check the error queue.

Timeouts

When using the API, the following timeouts should be used (or longer ones):

For what?	Example	Timeout (minimum)
General	`MEAS:INIT`	3 sec
Starting a measurement	`INIT START`	13 sec
Switching measurement functions	`MEAS:FUNC SLM`	5.5 sec

Command-List

Device Status

`*IDN?`

Shortcut	Identification: Reads the unique identification of the XL3.
Availability	always
Answer	<Manufacturer>,<Unit>,<Serial Number>,<FW Version>
	STR, STR, STR, STR
	Common command according to IEEE488.2-1992 10.14.3
Example	`*IDN?` `NTi Audio XL3 Control API, A3A-00129-B1, 0.90.4760`

`*CLS`

Shortcut	Executes a status reset: Clears the error queue and the output buffer.
Availability	always
Example	`*CLS` `LF`

*RST

Shortcut	Executes a device reset and should be the first command when starting a remote session to ensure that all XL3 settings make sense for remote measuring.
Availability	always
Example	`*RST` `LF`

Details	When using the Control API, execute this command first to avoid unwanted side effects.
	This command:
	• stops any running measurement
	• ends Measurement Series
	• exits any configuration
	• selects the Sound Level Meter function and sets the following:
	  • Logging interval: off
	  • Save mode: off
	  • Spectrum
	  • 1/3 Octave resolution
	  • LZF
	  • Gliding & Level Statistics to default values
	  • Timer mode: Continuous

INITiate Subsystem

`INITiate`

Shortcut	Starts or stops a measurement
Availability	Sound Level Meter
Parameter	[`START` | `STOP`]
	CHARDAT
Example	`INIT START`
Details	Time-dependent parameters like LAeq, LAFmax, etc., are undefined until START has been initiated. The start procedure may take a few seconds. When a measurement is stopped with STOP, the calculation of time-dependent parameters is stopped, and the result stays constant. If measurement series is enabled, the measurement can not be started or stopped. In this case, error 1048 is pushed to the error queue. ℹ For START, the answer LF is sent when the start procedure has finished. For STOP, any save dialog is suppressed.

`INITiate:STATe?`

Shortcut	Queries the run status of a measurement
Availability	always
Answer	[`STOPPED` | `FROZEN` | `SETTLING` | `RUNNING` | `PAUSED` | `UNDEFINED`]
	CHARDAT
Example	`INIT:STATE?` `RUNNING`

MEASure Subsystem

`MEASure:INITiate`

Shortcut	Triggers a measurement
Availability	always
Example	`MEAS:INIT`

Details	All measurement results of the MEASure subsystem are stored synchronously by this command. Before the first MEAS:INIT has been sent, all measurement values are undefined.
	
	A typical workflow is
	`INIT START`
	`MEAS:INIT`
	`MEAS:SLM:123? <para1>, <para2>`
	`MEAS:INIT`
	`MEAS:SLM:123? <para1>, <para2>`
	`...`

`MEASure:TIMer?`

Shortcut	Queries the actual measurement timer value.
Availability	Sound Level Meter
Answer	<timer> sec
	float UNIT
	0.1 seconds resolution (1 decimal)

Example	`MEAS:INIT`
	`MEAS:TIMER?`
	`3765.0 sec`

Details	This represents the time since initiating START

`MEASure:TIMer:MODE`

Shortcut	Defines the timer mode
Availability	Run state = STOPPED
Parameter	[`CONTinuous` | `SINGle` | `REPEated`]
	CHARDAT
Example	`MEAS:TIM:MODE CONT`
Details	

`MEASure:TIMer:MODE?`

Shortcut	Queries the timer mode
Availability	always
Answer	[`CONTINUOUS` | `SINGLE` | `REPEATED`]
	CHARDAT
Example	`MEAS:TIM:MODE?` `CONTINUOUS`

`MEASure:TIMer:SET:SINGle`

Shortcut	Sets the time value for the single mode
Availability	Run state = STOPPED
Parameter	<timer value>
	NUM_L Unsigned integer value in milliseconds

Example	`MEAS:TIM:SET:SING 30000`
Details	If the given value is outside of the valid range, the set value is corrected to the maximum or minimum, respectively. An error is returned if the given value is outside the range for a 32-bit unsigned integer.

`MEASure:TIMer:SET:SINGle?`

Shortcut	Queries the timer value for the single mode. The value is returned in milliseconds.
Availability	always
Answer	<timer value>
	NUM_L
Example	`MEAS:TIM:SET:SING?` `30000`

`MEASure:TIMer:SET:REPEated`

Shortcut	Sets the time value for the repeated mode
Availability	Run state = STOPPED
Parameter	<timer value>
	NUM_L Unsigned integer value in milliseconds
Example	`MEAS:TIM:SET:REPE 180000`
Details	If the given value is outside of the valid range, the set value is corrected to the maximum or minimum, respectively. An error is returned if the given value is outside the range for a 32-bit unsigned integer.

MEASure:TIMer:SET:REPEated?

Shortcut	Queries the timer value for the repeated mode. The value is returned in milliseconds.
Availability	always
Answer	<timer value>
	NUM_L
Example	`MEAS:TIM:SET:REPE?` `180000`

MEASure:FUNCtion

Shortcut	Defines the active measurement function
Availability	Run state = STOPPED
Parameter	[`SLM` | `RT` | `SI`]
	CHARDAT
Example	`MEAS:FUNC SLM`
Details	Switching between measurement functions may take 1-2 seconds. Leaves a possibly active configuration.
Restrictions	The SI parameter is only available if the Sound Insulation Option is installed

MEASure:FUNCtion?

Shortcut	Queries the active measurement function
Availability	always
Answer	[`SLM` | `RT` | `SI`]
	CHARDAT

Example	`MEAS:FUNC?` `SLM`

`MEASure:DECImals`

Shortcut	Selects the number of decimals of SLM wideband and spectrum results. This setting is not persistent and is reset to the default value LCD on device startup and kept until the device is reboot. The extended setting adds two additional decimals in the result.
Availability	always
Parameter	[LCD|EXTended]
	CHARDAT
Example	`MEAS:DECI EXT`

`MEASure:DECImals?`

Shortcut	Gets the decimals mode
Availability	always
Example	`MEAS:DECI?` `EXTENDED`

MEASure: SLM Subsystem

`MEASure:SLM:123?`

Shortcut	Queries a broadband measurement result of the Sound Level Meter.
Availability	Sound Level Meter
Parameter	[`LxS` | `LxSMAX` | `LxSMIN` | `LxF` | `LxFMAX` | `LxFMIN` | `LxEQ` | `Prev_LxEQ` | `LxPK` | `LxPKMAX` | `LyEQ_gt` | `LyEQ_gtMAX` | `LAFT3` | `LAFT3EQ` | `LAFT5` | `LAFT5EQ` | `LAFT5EQ-LAEQ` | `LCEQ-LAEQ` | `k1` | `k2`]
	CHARDAT
	x = [A|C|Z], y = [A|C] t = [5sec|10min|15min|60min] One of the four settings specified on the */Sound Level Meter/Gliding Leq Levels* page of the XL3 e.g. `LAEQ_g5sec` for `LAEQ_g5"` or `LCEQ_g15minMAX` or `LCEQ_g15'max`
Parameter (Additional with installed Extended Noise Measurement option)	[`LxI` | `LxIEQ` | `LxIMAX` | `LxIMIN` | `Ln%` | `LAIEQ-LAEQ`]
	CHARDAT
	x = [A|C|Z] n = [1|5|10|50|90|95|99] One of the seven statistic values specified on the */Sound Level Meter/Level Statistics* page of the XL3, e.g. L90.0% (if the decimal place is zero, you can also use L90%)
Answer	<Level> dB, [OK|UNDEF|LOW|OVLD]
	float UNIT CHARDAT

Example	`INIT START`
	`MEAS:INIT`
	`MEAS:SLM:123? LASMAX`
	`53.8 dB, OK`

Details	Returns a broadband result parameter stored by the last MEAS:INIT command. If an error occurs, e.g. the parameter is unknown, a ";" is returned. Statistic values: • For custom settings, use the custom values to read e.g. `MEAS:SLM:123? L33.3%` • Remotely changing/reading the settings is not implemented. • Be aware of the decimal separator. Use the Decimal Separator Configuration from the */System Settings/General* page. ℹ All parameter keywords are listed in their full keyword notation, even if some lowercase letters are used to describe special cases. No parameter keywords have abbreviations.

Call with multiple parameters	This command accepts up to 10 parameters. Each parameter has to be separated by a comma.

Example	`INIT START`
	`MEAS:INIT`
	`MEAS:SLM:123? LASMAX, LAFMAX, LZSMAX, LZFMAX`
	`52.1 dB, OK;54.8 dB, OK;6dB, OK;65.3 dB, OK`

Details	Because the line feed character (LF) is reserved as the message termination character, the responses for each parameter are separated by semicolons.
	If the query for a specific parameter generates an error, an empty field is returned, and the corresponding errors are pushed into the error queue. E.g. if L55% is not available:
	`MEAS:SLM:123? LASMAX, L55%, LAFMAX, L5%`
	`52.1 dB, OK;;54.8 dB, OK;`

`MEASure:SLM:123:DT?`

Shortcut	Queries a broadband dt measurement result of the Sound Level Meter.
Availability	Sound Level Meter
Parameter	[`LxSMAX` | `LxSMIN` | `LxFMAX` | `LxFMIN` | `LxEQ` | `LxPKMAX`]
	CHARDAT
	x = [A|C|Z]
Parameter (Additional with installed Extended Noise Measurement option)	[`LxIMAX` | `LxIMIN` | `LxE`]
	CHARDAT
	x = [A|C|Z]
Answer	<Level> dB, [OK|UNDEF|LOW|OVLD]
	float UNIT CHARDAT

Example	`INIT START`
	`MEAS:INIT`
	`MEAS:SLM:123:dt? LASMAX`
	`53.8 dB, OK`

Details	Queries a broadband result parameter of the Sound Level Meter stored with the last MEAS:INIT command. dt measurements are cleared after each MEAS:INIT, so this function returns the e.g. LEQ between two MEAS:INIT commands. The values have the same meaning as the dt values found in XL3 log files. If the parameter is unknown, a ";" is returned. All parameter keywords are listed in their full keyword notation, even if some lowercase letters are used to describe special cases. Parameter keywords don't have abbreviations.

Call with multiple parameters	This command accepts up to 10 parameters. Each parameter has to be separated by a comma.

Example	`INIT START`
	`MEAS:INIT`
	`MEAS:SLM:123:dt? LASMAX, LAFMAX, LZSMAX, LZFMAX`
	`52.1 dB, OK;54.8 dB, OK;6dB, OK;65.3 dB, OK`

Details	Because the line feed character (LF) is reserved as the message termination character, the responses for each parameter are separated by semicolons.
	If the query for a specific parameter generates an error, an empty field is returned e.g. for the command
	`MEAS:SLM:123:dt? LASMAX, LAIMAX, LAFMAX, LCIMAX`
	if the Extended Noise Measurement option is not installed, the following string is returned:
	`52.1 dB, OK;;63.7 dB, OK;`
	and the corresponding errors are pushed into the error queue.

`MEASure:SLM:SPECtrum?`

Alternative command keywords for XL2 compatibility: `MEASure:SLM:RTA?`

Shortcut	Queries the spectral results of the Sound Level Meter.
Availability	Sound Level Meter
Parameter	[`LIVE` | `MAX` | `MIN` | `EQ` | `CAPT` | `HOLD3`]
	CHARDAT
Parameter (Additional with installed Extended Noise Measurement option)	[`E` | `n%`]
	CHARDAT
	n = [1|5|10|50|90|95|99] One of the seven statistic values specified on the */Sound Level Meter/Level Statistics* page of the XL3, e.g. 10.0% (if the decimal place is zero, you can also use 10%)
Answer	{Level$_n$,} dB|dBu|dBV|V, [OK|UNDEF|OVLD]
	floats UNIT CHARDAT
	1/1 Oct: n = 12, f_{start} = 8 Hz 1/3 Oct: n = 36, f_{start} = 6.3 Hz Levels sorted from lowest to highest frequency

Example	`INIT START`
	`MEAS:INIT`
	`MEAS:SLM:SPEC? EQ`
	`46.3,50.7,34.5,45.4,42.2,37.2,39.0,39.8,32.1,28.5,29.8,31.0 dB, LOW`

Details	Queries the spectral results of the Sound Level Meter stored by the last MEAS:INIT command. If the parameter is unknown, a ";" is returned. The unit (dB, dBu, dBV, V) is adopted by the setting of the user interface.

`MEASure:SLM:SPECtrum:DT?`

Alternative command keywords for XL2 compatibility: `MEASure:SLM:RTA:DT?`

Shortcut	Queries the dt spectral results of the Sound Level Meter.
Availability	Sound Level Meter
Parameter	[`EQ`]
	CHARDAT
Parameter (Additional with installed Extended Noise Measurement option)	[`E`]
	CHARDAT
Answer	{$Level_n$,} dB|dBu|dBV|V, [OK|UNDEF|OVLD]
	floats UNIT CHARDAT
	1/1 Oct: n = 12, f_{start} = 8 Hz 1/3 Oct: n = 36, f_{start} = 6.3 Hz Levels sorted from lowest to highest frequency

Example	`INIT START`
	`MEAS:INIT`
	`MEAS:SLM:SPEC:DT? EQ`
	`46.3,50.7,34.5,45.4,42.2,37.2,39.0,39.8,32.1,28.5,29.8,31.0 dB, LOW`

Details	Queries the spectral results parameter of the Sound Level Meter that has been stored by the last MEAS:INIT command. dt measurements are cleared after each MEAS:INIT, so this function returns the LEQ of LE between two MEAS:INIT commands. The values have the same meaning as the dt values found in XL2 log files. If the parameter is unknown, a ";" is returned. The unit (dB, dBu, dBV, V) is adopted by the setting of the user interface.

`MEASure:SLM:SPECtrum:RESolution`

Alternative command keywords for XL2 compatibility: `MEASure:SLM:RTA:RESOlution`

Shortcut	Defines the resolution in which the RTA results are acquired.
Availability	Sound Level Meter with run state = STOPPED
Parameter	[`1/1` | `1/3` | `OCT` | `TERZ`]
	CHARDAT
Example	`MEAS:SLM:SPEC:RES 1/3`

`MEASure:SLM:SPECtrum:RESolution?`

Alternative command keywords for XL2 compatibility: `MEASure:SLM:RTA:RESOlution?`

Shortcut	Queries the resolution in which the RTA results are acquired.
Availability	Sound Level Meter
Answer	[`1/1` | `1/3`]
	CHARDAT
Example	`MEAS:SLM:SPEC:RES?` `1/3`
Details	The response of the alternative query returns the standard parameter keywords.

`MEASure:SLM:SPECtrum:WEIGhting`

Alternative command keywords for XL2 compatibility: `MEASure:SLM:RTA:WEIGhting`

Shortcut	Defines the frequency and time weighting in which the RTA results are acquired.
Availability	Sound Level Meter with run state = STOPPED
Parameter	[`AF` | `AS` | `CF` | `CS` | `ZF` | `ZS`]
	CHARDAT
Example	`MEAS:SLM:SPEC:WEIG CF`

`MEASure:SLM:SPECtrum:WEIGhting?`

Alternative command keywords for XL2 compatibility: `MEASure:SLM:RTA:WEIGhting?`

Shortcut	Queries the frequency and time weighting in which the RTA results are acquired.
Availability	Sound Level Meter
Answer	[`AF` | `AS` | `CF` | `CS` | `ZF` | `ZS`]
	CHARDAT
Example	`MEAS:SLM:SPEC:WEIG?` `ZF`
Details	The response of the alternative query returns the standard parameter keywords.

MEASure:RT60 Subsystem

`MEASure:RT60?`

Shortcut	Queries the average results of the RT60 analyzer
Availability	Reverberation time measurement function
Parameter	[`EDT` | `T15` | `T20` | `T30`]
	CHARDAT
Answer	<results>
	array of float 1/1 Oct: n = 8 1/3 Oct: n = 32
Example	`MEAS:RT60? T30` `0.18,0.16,0.31,0.18,0.39,0.41,0.36,0.33 sec, OK`
Details	This command returns the average results. If no measurement has been started, the undefined value is returned. An average result is available when at least a cycle is measured. After each cycle, the intermediate average result is returned, and when the measurement is terminated, the final average. The number of values in the array depends on the selected octave resolution.

`MEASure:RT60:ATTR?`

Shortcut	Queries the attributes of the average results of the RT60 analyzer
Availability	Reverberation time measurement function
Parameter	[`EDT` | `T15` | `T20` | `T30`]
	CHARDAT
Answer	<results>
	array of status characters 1/1 Oct: n = 8 1/3 Oct: n = 32
Example	`MEAS:RT60:ATTR? T30` `<,<,-,-,-,-,-,-`
Details	Each value in the values array has its status character in the second status array. The characters have the following meanings: – ok, undefined < decay too short > decay too long N low SNR D insufficient SNR ~ decay not linear E error

`MEASure:RT60:RESolution`

Shortcut	Defines the resolution in which the RT60 results are acquired.
Availability	always
Parameter	[`1/1` | `1/3`]
	CHARDAT
Example	`MEAS:RT60:RES 1/1`
Restrictions	The parameter 1/3 is only available if the Extended Room Acoustics Option is installed.

`MEASure:RT60:RESolution?`

Shortcut	Queries the resolution, in which the RT60 results are acquired.
Availability	always
Answer	[`1/1` | `1/3`]
	CHARDAT
Example	`MEAS:RT60:RES?` `1/1`

MEASure:RT60:TRIGger:LEVel:MINimum

Shortcut	Defines the minimum trigger level.
Availability	always
Parameter	<level>
	NUM_L
	Float value in dBSPL
Example	`MEAS:RT60:TRIG:LEV:MIN 80.0`
Details	The value must be in the range [-100 … +200]

MEASure:RT60:TRIGger:LEVel:MINimum?

Shortcut	Queries the minimum trigger level.
Availability	always
Answer	<level>
	NUM_L
	Float value in dBSPL
Example	`MEAS:RT60:TRIG:LEV:MIN?` `80.0`

MEASure:STIPA Subsystem

`MEASure:STIPA?`

Shortcut	Queries the results of the STIPA analyzer
Availability	STIPA measurement function
Answer	<measured STIPA value>,<STIPA value incl. noise>,<STIPA unit>,<status>,<rating text>,<rating letter>,<LAeq>,<LAS>,<run status>|<LZeq[125Hz … 8kHz]>|<MF1[125Hz … 8kHz]>,<MF2[125Hz … 8kHz]>|<status[125Hz … 8kHz]>
	NUM_L, NUM_L, UNIT, CHARDAT, CHARDAT, CHARDAT, NUM_L, NUM_L, CHARDAT| NUM_L, NUM_L, NUM_L, NUM_L, NUM_L, NUM_L, NUM_L| NUM_L, NUM_L, NUM_L, NUM_L, NUM_L, NUM_L, NUM_L| NUM_L, NUM_L, NUM_L, NUM_L, NUM_L, NUM_L, NUM_L| CHARDAT, CHARDAT, CHARDAT, CHARDAT, CHARDAT, CHARDAT, CHARDAT

Example	`MEAS:STIPA?`
	`TBD`
	`1.00,0.40,STI,FAULT,POOR,J,72.0,44.2,STOPPED|`
	`73.1,71.4,63.3,53.6,54.9,69.5,60.3|3.53,3.52,2.96,2.38,1.21,2.69,2.71`
	`|3.00,2.09,3.56,2.93,2.25,3.10,3.14|`
	`FAULT,FAULT,FAULT,FAULT,FAULT,FAULT,FAULT`

Details	This command returns the STIPA results. If no measurement has been started, the undefined value is returned.

`MEASure:STIPA:ANC:STATe`

Shortcut	Configures STIPA ambient noise correction setting.
Availability	Always
Parameter	[`OFF` | `ON`]
	CHARDAT
Example	`MEAS:STIPA:ANC:STAT ON`

`MEASure:STIPA:ANC:STATe?`

Shortcut	Queries the STIPA ambient noise correction setting.
Availability	always
Answer	[`OFF` | `ON`]
	CHARDAT
Example	`MEAS:STIPA:ANC:STAT?` `ON`

`MEASure:STIPA:ANC:SOURce?`

Shortcut	Queries the STIPA ambient noise correction source.
Availability	always
Answer	[`DEFAULT` | `MANUALLY_EDITED` | `MEASURED` | `IMPORTED_FROM_FILE`]
	CHARDAT
Example	`MEAS:STIPA:ANC:SOUR?` `MANUALLY_EDITED`

MEASure:STIPA:ANC:SPECtrum

Shortcut	Configures STIPA ambient noise correction spectrum.
Availability	always
Parameter	`<value 125Hz>,<value 250Hz>,<value 500Hz>,<value 1kHz>,<value 2kHz>,<value 4kHz>,<value 8kHz>`
	7 float values
Example	`MEAS:STIPA:ANC:SPEC 77.5,80.5,99.0,74.0,68.0,62.0,56.0`

MEASure:STIPA:ANC:SPECtrum?

Shortcut	Queries the STIPA ambient noise correction spectrum values.
Availability	always
Answer	`<value 125Hz>,<value 250Hz>,<value 500Hz>,<value 1kHz>,<value 2kHz>,<value 4kHz>,<value 8kHz>`
	7 float values
Example	`MEAS:STIPA:ANC:SPEC?` `77.5,80.5,99.0,74.0,68.0,62.0,56.0`

`MEASure:STIPA:EDITion`

Shortcut	Configures STIPA edition setting.
Availability	Always
Parameter	[`ED5` | `ED4` | `ED3` | `ED2`]
	CHARDAT
Example	`MEAS:STIPA:EDIT ED5`

`MEASure:STIPA:EDITion?`

Shortcut	Queries the STIPA edition setting.
Availability	always
Answer	[`ED5` | `ED4` | `ED3` | `ED2`]
	CHARDAT
Example	`MEAS:STIPA:EDIT?` `ED5`

`MEASure:STIPA:UNIT`

Shortcut	Configures STIPA unit setting.
Availability	Always
Parameter	[`STI` | `CIS`]
	CHARDAT
Example	`MEAS:STIPA:UNIT STI`

`MEASure:STIPA:UNIT?`

Shortcut	Queries the STIPA unit setting.
Availability	always
Answer	[`STI` | `CIS`]
	CHARDAT
Example	`MEAS:STIPA:UNIT?` `STI`

INPut Subsystem

`INPut:PHANtom`

Shortcut	Configures the phantom power setting.
Availability	When an ASD sensor is not connected.
Parameter	[`OFF` | `ON`]
	CHARDAT
Example	`INP:PHAN OFF`

`INPut:PHANtom?`

Shortcut	Queries the phantom power setting.
Availability	always
Answer	[`OFF` | `ON`]
	CHARDAT
Example	`INP:PHAN?` `off`

CALIbrate Subsystem

The abbreviation CALI instead of CAL was only defined for compatibility with XL2 commands.

`CALIbrate:MICrophone:TYPE?`

Shortcut	Queries the sensor type
Availability	always
Answer	STR
Example	`CALI:MIC:TYPE?` `M4260`
Details	If no ASD microphone is currently connected, the command returns *noASD*

`CALIbrate:MICrophone:SERIal?`

Shortcut	Queries the sensor serial number
Availability	always
Answer	STR
Example	`CALI:MIC:SERI?` `1234`
Details	If no ASD microphone is currently connected, the command returns *0*

`CALIbrate:MICrophone:SENSitivity:VALUe?`

Shortcut	Queries the microphone sensitivity in V/Pa
Availability	always
Answer	<sens> V/Pa, OK
	floats UNIT CHARDAT

Example	`CALI:MIC:SENS:VALU?` `20.0e-3 V/Pa, OK`

`CALIbrate:MICrophone:CIC`

Shortcut	Switches M2340 microphones (and MA230 mic preamps) into self-test mode. The microphone generates a reference tone (square wave with 31.25 Hz or 1 kHz), which the XL3 can measure.
Availability	always
Parameter	[`OFF` | `F1` | `F2`]
	CHARDAT
Example	`CALI:MIC:CIC OFF`
Details	To avoid CIC staying on (for example, if the remote connection breaks during CIC measurement), CIC turns off automatically after 60 seconds. If you need the CIC signal longer, retrigger the signal by sending the F1 or F2 command again within 60 seconds. This command will fail and return with an error code if no sensor is connected or the microphone doesn't support this operation.

`CALIbrate:MICrophone:TEMPerature?`

Shortcut	Queries the value of the integrated temperature sensor of the M2340 microphone.
Availability	always
Answer	<temperature> DEG_C, OK
	floats UNIT CHARDAT
Example	`CALI:MIC:TEMP?` `23.7 DEG_C, OK`
Details	This command will fail and return with an error code if no sensor is connected or the microphone doesn't support temperature readout.

SYSTem Subsystem

SYSTem:CONFiguration:JStream

Shortcut	Sends a configuration that is then written to the XL3
Availability	Run state = STOPPED
Parameter	JSON stream
	QSTR, embedded in single quotation marks only
Example	`SYST:CONF:JS '{"SLM":{"spectrum":{"octres":"1/3"}}}'`
Details	The transferred configuration may be the whole or partial configuration out of the complete configuration described in Available JSON Parameters (see page 79). The JSON stream must be embedded in single quotation marks because a JSON stream contains double quotation marks. Only the stated configuration values are changed if only a partial configuration is sent. The XL3 retains all other existing configuration values. If the JSON format is invalid, the command is rejected. The parameter may contain all values (characters) except the message termination character (LF, 0x0A).

SYSTem:CONFiguration:JStream?

Shortcut	Reads a complete system configuration and returns it as a JSON object
Availability	always
Answer	JSON stream
	TEXT
Example	`SYST:CONF:JS?` `{"SLM":{"spectrum":{"octres":"1/6","hold……`

Details	Note: The returned JSON stream is not embedded in single quotes as it is required for the command. The system configuration is described in Available JSON Parameters (see page 79).

SYSTem:CONFiguration:JStream:ERRor:TEXT?

Shortcut	Returns an additional error text if the last SYST:CONF:JS command has returned erroneously.
Availability	always
Answer	Single quoted error text
	QSTR

Example	`Assumed command for this example:`
	`syst:conf:js '{"SLM": {"spectrum": {, "hold_time": "3 sec", "fwtw": "ZF"}}}'`
	
	`SYST:CONF:JS:ERR:TEXT?`
	`'Missing a name for object member.|ectrum": {, "hold_time": "3 se'`

Details	The error text consists of two parts where the second may be empty: specific JSON parser error and JSON stream excerpt. A pipe character separates the two parts. The error text is empty if the last command was successful or never executed.

SYSTem:CONFiguration:JEncoded

Shortcut	Same as SYSTem:CONFiguration:Stream with the parameter encoded in base64
Availability	Run state = STOPPED
Parameter	BASE64 unquoted string
	STR
Example	`SYST:CONF:JE` `ewogICAgIlNMTSI6IHsKICAgICAgICAic3BlY3RydW0iOiB7CiAgICAgICAgIC……`

Details	see SYSTem:CONFiguration:Stream

`SYSTem:CONFiguration:JEncoded?`

Shortcut	Same as SYSTem:CONFiguration:Stream? with the answer encoded in base64
Availability	always
Answer	BASE64 unquoted string
	STR
Example	`SYST:CONF:JE?` `ewogICAgIlNMTSI6IHsKICAgICAgICAic3BlY3RydW0iOiB7CiAgICAgICAgIC……`
Details	The system configuration is described in Available JSON Parameters (see page 79).

Available JSON Parameters

The XL3 can be configured using JSON and the commands `SYSTem:CONFiguration:JStream`, `SYSTem:CONFiguration:JEncoded`, or `SYSTem:CONFiguration:FILE:LOAD`

A complete list of available configuration parameters can be found in the `/media/Configurations/documentation.txt` file. This file is automatically created when the XL3 starts up.

`SYSTem:CONFiguration:FILE:LOAD`

Shortcut	Instructs the XL3 to load a configuration into the XL3 from a specified file
Availability	Run state = STOPPED
Parameter	file name
	QSTR
Example	`SYST:CONF:FILE:LOAD “<filename>”`
Details	The file `/media/Configurations/<filename>.xl3cfg` must exist and follow the Configuration File Format (see page 79). You may upload the file using the `SYSTem:FILE:WRITe` command.

Configuration File Format

A configuration file must have the extension *.xl3cfg*.

The configuration file consists of a header and a configuration data part. These two parts are divided by the separator row *##CONFIG:*

Each part is JSON formatted.

”MeasurementID”: [SLM|RT|SI] instructs the XL3 which screen to show

- SLM - Sound Level Meter
- RT - Reverberation Time
- SI - Sound Insulation

”UUID”: a Universal Unique Identifier

Example content of a <filename>.xl3cfg:

```
{
"MeasurementID": "SLM",
"UUID": "00000000-0000-0000-0000-000000000000"
}
##CONFIG:
{"SLM":{"spectrum":{"octres":"1/3"}}}
```

SYSTem:CONFiguration:DOCumentation

Shortcut	Creates media/Configurations/documentation.txt, which contains all configuration options available with the current FW
Availability	always
Example	`SYST:CONF:DOC`
Details	documentation.txt is formatted as JSON pretty

SYSTem:FILE:WRITe

Shortcut	Writes the data to the file given by its pathname
Availability	always
Parameter	<file_pathname>, <base64 encoded data>
	QSTR, STR
Example	`SYST:FILE:WRIT “<file_pathname>”, ewogICAgIk1lYXN1cmVtZW50SUQiOiAiU0xNIiwK...`

Details	The relative file pathname is extended to the full pathname /media/<file_pathname>. The folder separators may be slashes ('/') or backslashes ('\'). The first folder in the <file_pathname> (e.g. <SdCard1> or <Configurations>) must exist. Note that the names are case-sensitive. All following subfolders and the file are created if they don't exist. The filename must include the desired extension. An existing file will be overwritten with the new data. The binary file data must be passed as a base64 encoded string. The length of the entire command must not exceed 16k characters. This limits the file size to 12k bytes.

`SYSTem:FILE:READ?`

Shortcut	Reads the data from a file given by its pathname and returns it as base64 encoded string
Availability	always
Parameter	<file_pathname>
	QSTR
Answer	<base64 encoded data>
	STR
Example	`SYST:FILE:READ? "<file_pathname>"`
Details	The relative file pathname is extended to the full pathname /media/<file_pathname>. The folder separators may be slashes ('/') or backslashes ('\'). The file \media\<file_pathname> must exist. Note that the names are case-sensitive. The maximum length of the file to be read out is 12k bytes. Reading larger files will generate an error.

`SYSTem:POWer:SOURce?`

Shortcut	Reads the source of the power
Availability	always
Answer	[`NONE` | `USB` | `WALL_ADAPTER`]
	CHARDAT

Example	`SYST:POW:SOUR?` `WALL_ADAPTER`
Details	If the XL3 is powered with both a wall adapter and a USB power supply, the wall adapter is returned by this query.

`SYSTem:SOH?`

Shortcut	Reads the given State of Health items
Availability	always
Parameter	[`TIMESTAMP` | `VDCIN` | `BATTERYSOC` | `RUNSTATUS` | `WEATHERSTATIONS` | `FREESTORAGE` | `GPSLOCATION` | `IPHANTOM` | `LOCALTIME` | `TIMEZONE` | `TEMPERATURE` | `AIRPRESSURE` | `POWERSOURCE` | `CLOCKSOURCE` | `CPUTEMPERATURE` | `CICTIME` | `CICSTATUS` | `CICDELTAF1` | `CICDELTAF2` | `NOISELOCATOR`]
	CHARDAT
Example	`SYST:SOH? LOCALTIME,GPSLOCATION` `2024-02-09 14:57:20;47.174349 9.513426`
Details	Returned items are separated by semicolons if more than one is requested. Item names (parameters) are equal to these used for streaming. Also the format of the returned items is equal to the streaming.
Restrictions	The maximum of parameters is 10. So, to query all items two separate commands are required.

`SYSTem:OPTIons?`

Shortcut	Reads the installed options
Availability	always
Answer	<options list>

	list of CHARDAT
Example	`SYST:OPTI?` `FO, EN, ER, IN, NC, AP, DX, RA, SI, LW`
Details	The list contains the acronyms for the options: • FO - Full Option Pack • EB - Environmental Noise Bundle • IB - Sound Insulation Bundle • RB - Room Acoustics Bundle • EN - Extended Noise Measurement • ER - Extended Room Acoustics • IN - Sound Insulation • NC - NTi Connect Open Data • AP - API (Programming Interface) • DX - Data Explorer SW • RA - Room Acoustics Reporter SW • SI - Sound Insulation Reporter SW • LW - Sound Power Reporter SW

SYSTem:ERRor?

Shortcut	Queries the error queue
Availability	always
Answer	{errno$_n$,}
	NUM_L
	n 50

Example	`SYST:ERR?`
	`40, 70`
	`SYST:ERR?`
	`0`

Example with text	`SYST:ERR?`
	`40|"Wrong type of parameter(s)", 70|"Command keywords were not recognized"`
	`SYST:ERR?`
	`0|"No error"`

Details	XL3 Error codes differ from XL2 Error codes There are different classes of errors. Some errors refer to the command syntax, others to internal states of the XL3. Refer to the Error List (see page 86) below Every error is pushed into the error queue that must be queried to get the error number. Additionally an error text output can be enabled by the command `system:error:text` (see below). Error texts are embedded in double quotation marks and separated by the pipe character from its error number (see example above). Several error numbers from unused parser features may be returned. These numbers are not listed here.

`SYSTem:ERRor:TEXT`

Shortcut	Enables or disables text output for the `system:error?` query
Availability	always
Parameter	[`OFF` | `ON`]
	CHARDAT
Example	`SYST:ERR:TEXT ON`
Details	Error texts are initially disabled. After enabling error texts, error texts are enabled as long the Control API connection is active or it's disabled by command. When the connection is closed error texts are disabled automatically. Error texts are output together with their error numbers. See example for the error query above.

`SYSTem:ERRor:TEXT?`

Shortcut	Reads the current error text output status
Availability	always
Answer	[`OFF` | `ON`]
	CHARDAT

Example	`SYST:ERR:TEXT?` `ON`
Details	The default setting is OFF (disabled) and it must be enabled at every time a connection is opened.

Appendix

Error List

Number	Description
0	no error (queue is empty)
10	No input command to parse
14	Numeric suffix is an invalid value
20	Parameter of type Numeric Value overflowed its storage
30	Wrong units for parameter
40	Wrong type of parameter(s)
41	Wrong format of a type of parameter(s)
42	Invalid value of parameter(s)
50	Wrong number of parameters
60	Unmatched quotation mark (single/double) in parameters
65	Unmatched bracket
70	Command keywords were not recognized
75	Data block declaration is invalid, e.g. given length is larger than the size of the provided data
80	Item numeric insertion is an invalid value
82	DESIGN ERROR: Too many item numeric insertions in parameter Spec
83	No numeric insertion value found in the input item
85	Numeric insertion value expected by the translator
86	Value in numeric insertion must be positive (unexpected negative sign)
87	Value in numeric insertion must be integer value (no decimal separator / floating point)
88	Numeric insertion value is out of range

Number	Description
90	No numeric suffix value but expected
91	Numeric suffix value is out of range
150	Invalid Date Time Format
151	Content of Date Time input is invalid
200	IMPLEMENTATION ERROR: Called translator doesn't fit to the declared parameter list in the specs
201	IMPLEMENTATION ERROR: Called translator doesn't fit to the specific parameter declaration
202	IMPLEMENTATION ERROR: Parameter index is out of range
203	DESIGN ERROR: At least the first parameter must be defined as required in the param spec
204	IMPLEMENTATION ERROR: Requested parameter is not implemented in the local query function
300	Timeout while waiting for response message from core
301	Timeout while waiting for response message from core (configuration channel)
302	Timeout while waiting for response message from core (configuration channel)
303	Timeout while waiting for mirror update (configuration channel)
310	Requested broadband signal is not available (gliding eq or percentile)
311	Requested spectral signal is not available (percentile)
350	Input verification: The ASD Hex-stream is not valid
351	Input verification: The JSON input stream is not valid (see JSON parser error for details)
352	Input verification: The BASE64 input stream is not valid
353	Input verification: The JSON input stream is empty
370	Invalid configuration file header
371	Unexpected empty configuration file header
400	IMPLEMENTATION ERROR: No command handler specified for this command

Number	Description
401	Access to uninitialized mirror data
402	Invalid parameter passed to command handler (e.g. unexpected value for query)
403	Failed to translate parsed command parameter from the ParamSpec list into internal enumeration type
410	DESIGN ERROR: Dynamic downcast to branch/leaf failed
411	IMPLEMENTATION ERROR: Invalid parameter in deserialization (should never occur)
420	Input verification: The JSON input stream is not a valid object
450	API option required to executed this command
550	RUNTIME ERROR: File I/O error
551	RUNTIME ERROR: File buffer memory error
552	RUNTIME ERROR: File I/O: unexpected content
560	Value input verification: Invalid enum value (not in list)
561	Value input verification: Enum type is not text
562	Value input verification: Float type is not a number
563	Value input verification: Signed integer type is not a number
564	Value input verification: Unsigned integer type is not a number
565	Value input verification: Value is not a text
580	Invalid path or file name
581	Path is not relative (starts with ‘/' or '\’)
582	Path contains inexistent drive or media
583	Relative path is too long (exceeds internal buffer size)
584	Failed to create path
585	File I/O: Failed to create file for write operations
586	Input is an invalid base64 encoding string

Number	Description
587	File I/O: Error while writing data to file
588	Specified file not found
589	File is too large for read operation
590	File I/O: Failed to open file for read operations
591	OS call has unexpectedly failed
600	Failure while handling response message received from the core
601	Message received from the core does not contain valid RT data
700	WARNING: Not all elements in configuration were handled (unknown or misspellt), configuration was loaded
750	IMPLEMENTATION ERROR: Not further specified error during execution (parsing, translation, handling)
751	IMPLEMENTATION ERROR: Exception during error translation (exception due to unknown error, recompilation with new generated code required)
800	Error queue overflow
810	Input buffer overflow
970	Program Lost Error
1001	Value is out of range
1002	Command rejected: Measurement is running
1003	Not implemented
1004	Parameter is not available
1005	Request forbidden
1006	Internal memory error
1007	ASD device not present
1008	ASD page index out of valid range
1009	ASD operation failed

Number	Description
1010	License required
1011	(internal error) Attempt to load configuration parameter while not in loading state
1012	Configuration contains incomplete SLM limit setup
1013	Configuration file I/O error
1014	Memory error while processing configuration file
1015	Configuration file format is invalid
1016	Configuration file header is invalid
1017	Failed to parse configuration
1018	Internal error: Timeout while waiting for load response
1019	Memory error while loading configuration
1020	Internal error: Unknown error
1021	Internal error: Error not further specified
1022	The specified audio sample rate for recording is not allowed for compressed format
1023	Configuration load error: Specified Enum parameter not found
1024	Configuration load error: Enum type mismatch
1025	Configuration load error: Float type mismatch
1026	Configuration load error: Signed Int type mismatch
1027	Configuration load error: Unsigned Int type mismatch
1028	Configuration load error: Text type mismatch
1029	Warning: The configuration file contains unknown entries
1030	Configuration load error: Selected accessory doesn't fit to the connected microphone
1031	Configuration load error: Selected diffuse field correction doesn't fit to the connected microphone
1032	The connected sensor (microphone) doesn't support the current operation

Number	Description
1033	There is no sensor (microphone) connected
1034	The current sensor (microphone) operation has failed
1035	Measurement Series is active
1036	Microphone is disconnected
1037	The preamplifier is not supported
1038	Unexpected change of preamplifier
1039	The preamplifier doesn't fit to the data from NTi server
1040	Updating Mic Model Number failed
1041	Updating Capsule Type failed
1042	Updating Microphone Sensitivity failed
1043	Unexpected change of the connected adapter
1044	Update to CS011 model number not allowed for this adapter
1045	This adapter model doesn’t support writing a new serial number
1046	Failed to update the serial number
1047	Measurement function is not Reverberation Time
1048	Measurement Series is enabled
1049	Loading configuration failed: File not found
1050	Loading configuration failed: Local operation active
1051	Set system date/time not allowed
1052	Failed to set system date/time
1053	Setting not supported
1054	Name contains illegal characters
1055	Audio recording permanently disabled

Number	Description
1056	Noise Locator not present
1057	Noise Locator operation failed
1058	Error while executing Noise Locator operation
10000	Streaming: No data found
10001	Streaming: No requested signals available
10002	Streaming: Too many signals
10008	Streaming: Internal error (with specific error text)
10009	Streaming: Not further specified internal error
10010	History streaming: File access error
10011	History streaming: Failed to open file
10012	History streaming: Failed to close file
10013	History streaming: File error, failed to set position (file or cache)
10014	History streaming: File error, failed to set position (file or cache)
10015	History streaming: Attempt to set a file position out of valid range
10016	History streaming: Attempt to use an invalid file type
10017	History streaming: Attempt to read from an invalid file position
10018	History streaming: File header contains invalid data
10019	History streaming: (Armed) Stream closed due to log file error

List of symbols

List of symbols used in the command description.

Symbol	Description
:	Colons separate keywords of an XL3 command.
[ ]	Square brackets enclose the *list of available parameters*, of which one must be selected.
|	A vertical line reads as a logical „OR“, i.e. this sign separates *alternative* parameters.
< >	Triangle brackets enclose the *variable parameters* that must be set for a user-defined value.
{ }	Braces have the same meaning as triangle brackets, except that the enclosed parameters can be included *several* times.
,	Commas separate arguments in an arguments list.
?	The question mark indicates a *query*.
( )	Round brackets enclose comments.

Parameter Types

Type	Description
STR	Unquoted String
QSTR	String embedded in single or double quotation marks
CHARDAT	Character Data
UNIT	Unit character data
NUM_L	Numerical value: Integer or Float. With optional unit
BIN	IEEE488 Binary Data
JSON	An open standard file format

Differences between XL3 and XL2

Difference	XL2	XL3
Line ending	“CR LF”	“LF”
Response to set commands (Command without “?”)	no	“LF” after command completion of set command
Multiple Commands separated by semicolons	no	yes
Device access	Only local PC via USB virtual COM port	Through a TCP (see page 10) socket or a WebSocket (see page 13) or a browser through NTi Connect (see page 9)
Command Keywords	Short or any variant of the full form	Short or complete long keyword is required
Channel open	No response	On channel open, the XL3 sends the Channel Identification Message or a *Password:* prompt
Symbols for second and minute in parameters	“, ‘	Symbols are replaced with words “sec” and “min”

Getting Started with Python and PyCharm Community Edition

This appendix is designed to help new Python users get productive quickly by using PyCharm Community Edition as their integrated development environment (IDE). Follow these steps to open an existing project, install the latest Python version, and add the websocket-client package required by the examples.

Installing the Latest Python Version

- Visit the Python official website[1].
- Download the latest version of Python suitable for your operating system.
- Run the installer and ensure that "Add Python to PATH" is selected before clicking "Install Now."

Installing PyCharm Community Edition

- Visit the PyCharm download page.
- Choose the Community Edition and download the installer for your operating system.
- Run the installer and follow the on-screen instructions to complete the installation.

Opening an Existing Project in PyCharm

- Open PyCharm from your applications menu or desktop shortcut.
- On the welcome screen, click on "Open."
- Navigate to the directory where your example code is stored.
- Select the folder containing the project files and click "OK."

Configure the Project Interpreter:

- Once the project is loaded, go to `File > Settings` (or `PyCharm > Preferences` on macOS).
- In the left sidebar, navigate to `Project: <project_name> > Python Interpreter`.
- Click the gear icon next to the interpreter dropdown and select "Add."
- Choose "System Interpreter" and select the newly installed Python version from the list.
- Click "OK" to apply the changes.

Adding the websocket-client Package

- Go to `File > Settings` (or `PyCharm > Preferences` on macOS).
- Install the Package:
  - In the left sidebar, navigate to `Project: <project_name> > Python Interpreter`.
  - Click the `+` button on the right side of the window to open the available packages window.
  - In the search bar, type websocket-client.
  - Select websocket-client from the list and click "Install Package."
- Verify the Installation:
  - Ensure the package appears in the list of installed packages under your project interpreter.

1. https://www.python.org/downloads/

Running the Example Code

Run a Python Script:

- In the Project Explorer pane, navigate to the Python script you want to run.
- Right-click the script file and select `Run '<script_name>'`.

Debugging:

- To debug the script, right-click the script file and select `Debug '<script_name>'`.
- Use breakpoints and the debugging tools provided by PyCharm to troubleshoot and inspect your code.