RealTime OpenControl Documentation

Release 2.0b0

Sebastian Keller

Jun 18, 2019

Contents

CHAPTER 1

Welcome to RealTime OpenControl's documentation

RealTime OpenControl (RTOC) is a free software for recording measurement data of various measuring instruments. It offers the following components

1.1 RTLogger backend

The core element of RTOC. More information here: *Backend Source-code*

1.2 Expandable with plugins

You can write plugins by your own or use a plugin from the repository. More information here: *Collecting data*

1.3 Telegram-Bot

The Telegram-Bot offers access to RTOC from any device with Telegram installed. More information here: *Telegram Communication*

1.4 TCP-Server

Communication with RTLogger from other processes or devices. Suitable for embedded devices with graphical user interface. More information here: *TCP Communication*

1.5 Webserver

View plots and events from any network device. More information here: *Run webserver*

1.6 Graphical user interface (RTOC-GUI)

Used, when running RTOC on computers/laptops to view and edit data. More information here *Graphical user interface*.

1.7 Scripting/Automation

You can write scripts and run/edit and stop them during runtime. You have full access to all data stored in RTOC and access to all plugins. A event/action system gives a simple solution for very custom automisations. More information here: *Controlling and automation*

CHAPTER 2

Getting started

Follow one of the installation-instructions (pip, builds, source): *Installation*

Have a look at the plugin-documentation: *Collecting data*

RTOC will create a directory in the home-directory, where all user-data is stored. More information here: *Userdata*

CHAPTER 3

FAQ

- How can I get plugins from the community? *Plugin repository*
- How do I import new data from CSV, Wave, Excel, ODF, Matlab? *Import/Export signals/sessions*
- How do I connect a new plugin? *Collecting data*
- How do I create a sub-GUI for a device? *Writing Plugins*
- How do I create my first script? *Controlling and automation*
- What does the trigger mean? *Trigger-System*
- RTOC library and default functions for scripts: `RTLogger.scriptLibrary`
- Can I access the data from any device? *Telegram Communication* or *TCP Communication*
- How do I use the graphical user interface? *Graphical user interface*
- How do I create a telegram bot? *Telegram-Bot setup*
- How do I control an RTOC server via TCP in the network? *Remote-control via TCP*
- Where can I find examples for plugins? RTOC repository

Feel free to buy me some coffee with milk

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

CHAPTER 5

Table of contents

5.1 Installation

Python3 needs to be installed on your target-computer. Download python3 from the official website: www.python.org

After installing python3, you are able to run python in a terminal. (`cmd.exe` on windows) .. code-block:: bash

    $ python3 –version Python 3.6.3

5.1.1 Installing with Python3 (recommended)

RTOC is available in the Python package manager PIP:

```
pip3 install RTOC
```

This will download the basic RTOC without the dependencies needed for the GUI, Telegram and the Webserver. The default RTOC setup is suitable for running RTOC on embedded devices.

There are also different variations available to install:

```
pip3 install RTOC[Webserver]    # Includes all packages for webserver
pip3 install RTOC[GUI]          # Includes all packages for GUI
pip3 install RTOC[Telegram]     # Includes all packages for Telegram
pip3 install RTOC[ALL]          # Includes all packages
```

5.1.2 Installing from builds

Download the latest release builds for Windows here.

Extract the .zip file into a directory. RTOC is started by double-clicking on “RTOC.exe”. Alternatively via command line:

```
// local RTOC-instance including GUI
./RTOC
```

5.1.3 Install manually

To use the basic RTOC, the following dependencies must be installed:

```
pip3 install numpy pycryptodomex requests python-nmap whaaaaat prompt_toolkit psycopg2
```

If you want to use the GUI you must also install the following packages:

```
pip3 install pyqt5 pyqtgraph markdown2 xlsxwriter scipy pandas ezodf pyGithub
```

If you want full functionality, then you still need the following packages (for telegram bot and webserver):

```
pip3 install python-telegram-bot matplotlib dash gevent dash_daq dash_table plotly␣
↪flask
```

You can use different stylesheets for the GUI if you want. Just install one of these with pip: 'QDarkStyle', 'qtmodern', 'qdarkgraystyle'.

The RTOC repository can then be cloned with:

```
git clone git@github.com:Haschtl/RealTimeOpenControl.git
```

5.1.4 Long-time measurements in postgreSQL database (optional)

If you want to store measurements for a long period of time, I would recommend to use RTOC with a postgreSQL database. Therefore you need to setup postgreSQL on your system and change the postgresql parameters in your *config.json* file.

Setup postgreSQL on linux

1. Open a terminal window
2. Issue the command `sudo apt install postgresql`
3. Follow the instructions to change the default postgresql-password.
4. Add a new user. You need to switch to the root user to create a new postgres-user

```
$ sudo bash
$ su - postgres
$ createuser --interactive --pwprompt
$ Enter name of role to add: <NEWUSERNAME>
$ Enter password for new role: <PASSWORD>
$ Enter it again: <PASSWORD>
$ Shall the new role be a superuser? (y/n) n
$ Shall the new role be allowed to create databases? (y/n) y
$ Shall the new role be allowed to create more new roles? (y/n) n
$ exit  // to return to root user
$ exit // to return to your user
```

5. Create a database

```
$ createdb -O <NEWUSERNAME> <DATABASE_NAME>
```

6. Enter your postgresql username, port, database and password in your *config.json* file.

Setup postgreSQL on windows

2. Download postgresql one-click installer from this website
3. Double click on the downloaded file and follow the setup instructions.
4. Add a new user and create a database (google for that)
5. Enter your postgresql username, port, database and password in your *config.json* file.

5.2 First steps

5.2.1 First run

After installing RTOC, you can run it with:

```
// local RTOC-instance including GUI
python3 -m RTOC

// local RTOC-instance without GUI (only TCP-Server, [HTTP-Server, Telegram-Bot])
// I would recommend starting RTOC as a service and not to use this.
python3 -m RTOC.RTLogger -s start/stop

// local RTOC-Configuration from Console
python3 -m RTOC.RTLogger -c

// local RTOC-Webserver at port 8050
python3 -m RTOC.RTLogger -w

// remote RTOC-instance with GUI
python3 -m RTOC -r <ADRESS>

// explicit local RTOC GUI (even if database is enabled)
python3 -m RTOC -l
```

5.3 Collecting data

5.3.1 Writing Plugins

To integrate a new device, you have to program your own plugin.

Each user-plugin must be stored in the user-directory `~/.RTOC/devices/`. Each plugin must has its own folder.

Plugins are written in Python3 and need to follow some rules:

```
from LoggerPlugin import LoggerPlugin # contains all plugin-functions

class Plugin(LoggerPlugin):  # This must be the main class of your function
      def __init__(self, *args, **kwargs):
        super(Plugin, self).__init__(*args, **kwargs))
```

(continues on next page)

(continued from previous page)

```
        # start your code here
```

The above lines **must** be copied exactly into the plugin!

All functions and parameters defined in the main class are available for scripts. To prevent them from being displayed in the Script Help (and Telegram-bot), parameters and functions must begin with '_'.

Refer to `LoggerPlugin` for a full list of available functions.

The following example plugin sends data to RTOC every second and loads an empty GUI

5.3.2 Template.py

```
"""
This template shows, how to implement plugins in RTOC

RTOC version 2.0

A plugin needs to import RTOC.LoggerPlugin to be recognized by RTOC.
"""
try:
    from LoggerPlugin import LoggerPlugin
except ImportError:
    from RTOC.LoggerPlugin import LoggerPlugin

import sys
import time
from PyQt5 import uic
from PyQt5 import QtWidgets
import numpy as np

DEVICENAME = "Template"
"""Definition of the devicename outside of plugin-class"""

AUTORUN = True
"""If true, the thread to collect data will run right after initializing this plugin""
↪"
SAMPLERATE = 1
"""The thread,which is supposed to collect data will be executed with 1 Hz"""


class Plugin(LoggerPlugin):
    def __init__(self, *args, **kwargs):
        super(Plugin, self).__init__(*args, **kwargs)
        """Call this to initialize RTOC.LoggerPlugin"""

        self.setDeviceName(DEVICENAME)
        """
        Set a default devicename.
        This will be used for all signals sent by this plugin as default
        """

        self.smallGUI = True
        """This plugin has a GUI, which is small. GUI can be shown in two different␣
↪ways."""
```

(continues on next page)

(continued from previous page)

```
        self._firstrun = True

        self.setPerpetualTimer(self._updateT, samplerate=SAMPLERATE)
        """You will need to collect data periodically in many applications. You need␣
↪to start that in a seperate thread.

        RTOC provides a simple way to start a repeated thread with :py:meth:`.
↪LoggerPlugin.setPerpetualTimer`. The first parameter is the function, which␣
↪collects data and sends it to RTOC. You can define a ``samplerate`` or an␣
↪``interval`` to set the samplerate.

        You can still use normal threads to do the same thing, but in this way, the␣
↪plugin can be stopped properly. If you are using normal threads, make sure, to have␣
↪a loop limited by '
        ``self.run`` with ``while self.run:``.
        """

        if AUTORUN:
            self.start()
            """
            Start the configured perpetualTimer. You can stop it with ``self.
↪cancel()``. If you want to start data collection, when this plugin is started, you␣
↪need to call ``self.start()`` in the plugin-initialization.
            """

    def _updateT(self):
        """
        This function is called periodically after calling ``self.start()``.

        This example will generate a sinus and a cosinus curve. And send them to RTOC.


        """
        y1 = np.sin(time.time())
        y2 = np.cos(time.time())

        self.stream([y1, y2], snames=['Sinus', 'Cosinus'], unit=["kg", "m"])
        """
        Use this function to send data to RTOC: :py:meth:`.LoggerPlugin.stream`
        """

        self.plot([-10, 0], [2, 1], sname='Plot', unit='Wow')
        """
        Use this function to send data to RTOC: :py:meth:`.LoggerPlugin.plot`
        """
        if self._firstrun:
            self.event('Test event', sname='Plot', id='testID')
            """
            Use this function to send an event to RTOC: :py:meth:`.LoggerPlugin.event`
            """
            self._firstrun = False

    def loadGUI(self):
        """
        This function is used to initialize the Plugin-GUI, which will be available␣
↪in :doc:`GUI`.
```

(continues on next page)

(continued from previous page)

```
        This is optional.

        Returns:
            PyQt5.QWidget: A widget containing optional plugin-GUI
        """
        self.widget = QtWidgets.QWidget()
        """
        Create an empty QWidget
        """
        packagedir = self.getDir(__file__)
        """Get filepath of this file"""
        uic.loadUi(packagedir+"/Template/template.ui", self.widget)
        """
        This example will load a QWidget designed with QDesigner
        """
        self.widget.teleMessageButton.clicked.connect(self._teleMessageAction)
        self.widget.telePhotoButton.clicked.connect(self._telePhotoAction)
        self.widget.teleFileButton.clicked.connect(self._teleFileAction)
        """
        Connect GUI-buttons with python-functions
        """
        return self.widget  # This function needs to return a QWidget

    def _teleMessageAction(self):
        text = 'Hello world!'
        self.telegram_send_message(text, onlyAdmin=False)

    def _telePhotoAction(self):
        path = self.getDir(__file__)+'/examplePhoto.png'
        self.telegram_send_photo(path, onlyAdmin=False)

    def _teleFileAction(self):
        path = self.getDir(__file__)+'/examplePhoto.png'
        self.telegram_send_document(path, onlyAdmin=False)



hasGUI = True  # If your plugin has a widget do this

if __name__ == "__main__":
    """
    Sometimes you want to use plugins standalone also. This is very useful for␣
↪testing.
    """
    if hasGUI:
        app = QtWidgets.QApplication(sys.argv)
        myapp = QtWidgets.QMainWindow()

    widget = Plugin()

    if hasGUI:
        widget.loadGUI()
        myapp.setCentralWidget(widget.widget)

        myapp.show()
        app.exec_()
```

(continues on next page)

(continued from previous page)

```
    widget.run = False

    sys.exit()
```

5.3.3 Plugin repository

This repository contains some plugins for RealTime OpenControl (RTOC).

5.3.3.1 Installing plugins

You can either install plugins manually or - if you are using the GUI - you can install it with the built-in installer: *Plugin-Downloader*

5.3.3.1.1 Manually

To add a plugin to RTOC you need to do the following steps:

1. Install RTOC (*pip3 install RTOC*) - You will need to run RTOC once
2. Copy the folder of the desired plugin to your RTOC-Userpath: *~/.RTOC/devices/*
3. Now restart RTOC (*python3 -m RTOC*)

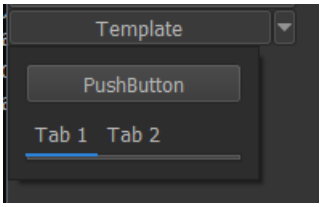
5.3.3.2 List of plugins

- Template: An example, showing how to create a simple plugin to send data to RTOC
- DPS5020: Plugin for DPS powersupplies. It can monitor all data and you can set Voltage, Current and switch it on/off. Uses USB to read data.
- Funktionsgenerator: Default-plugin of RTOC. Generates common signals.
- holdPeak_VC820: Plugin for VC820 multimeters. It can monitor the measured values with correct units. Uses USB/Minimalmodbus to read data.
- INA219_Modul: Plugin for INA219 solar module. Monitors voltage, current, power and shunt_voltage
- Octotouch: Plugin for 3D-printer-software Octotouch. It can just monitor the temperatures. Uses HTTP/JSON to read data.
- PIKO_Solarmodules: Plugin for PIKO solar modules. Monitors voltage, current and power
- System: Plugin to monitor system-information like CPU, Memory, ...
- ReflowOfen/ReflowPlatte: Plugin, which reads data from local network-devices HTTP-address.
- Heliotherm: Plugin, which reads data from Heliotherm heat pump using TCP/Modbus.
- Futtertrocknung: Embedded-Plugin. Plugin, which is used to run on a RaspberryPi. Monitors some sensor-data.

5.3.3.3 Plugin descriptions

5.3.3.3.1 Template

GUI: Yes, if you want to



Dependencies: -

Info: Use this plugin as a starting point

5.3.3.3.2 DPS5020

GUI: Yes

Dependencies: *pip3 install minimalmodbus*

Target system: Each OS (connected to DPS with USB)

Info:

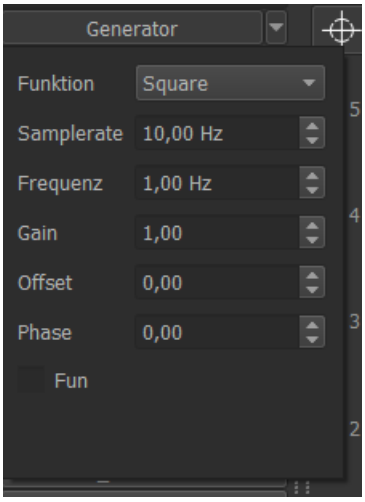
- You can set a parameters in file DPS5020.py

```
default_device = '/dev/ttyUSB0'
SERIAL_BAUDRATE = 9600
SERIAL_BYTESIZE = 8
SERIAL_TIMEOUT = 2
```

- You will need to run RTOC as root unless you set devices rules. See [this tutorial](http://ask.xmodulo.com/change-usb-device-permission-linux.html) for how to set device rules.

5.3.3.3.3 Generator

GUI: Yes



Dependencies: -

Target system: Each OS

Info:

5.3.3.3.4 holdPeak_VC820

GUI: Yes

Dependencies: *pip3 install serial*

Target system: Each OS (connected to VC820 with USB)

Info:

- You can set a parameters in file HoldPeak VC820.py: - default_device = 'COM7' - SERIAL_BAUDRATE = 2400 - SERIAL_BYTESIZE = 8 - SERIAL_TIMEOUT = 1
- You will need to run RTOC as root unless you set devices rules. See [this tutorial](http://ask.xmodulo.com/change-usb-device-permission-linux.html) for how to set device rules.

5.3.3.3.5 INA219_Modul

GUI: No

Dependencies: *pip3 install pi-ina219*

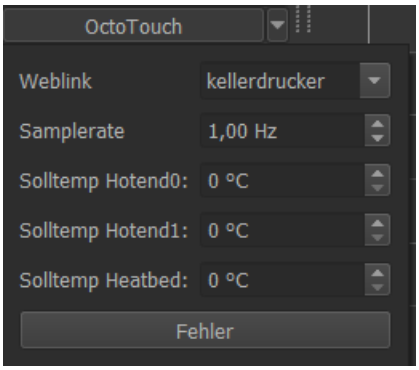
Target system: RaspberryPi (connected to INA219 via I2C)

Info:

- You can set a parameters in file INA219_Modul.py:
  - SHUNT_OHMS = 0.1
  - MAX_EXPECTED_AMPS = 0.2
  - SAMPLERATE = 1/60# frequency in Hz (1/sec)
  - I2C_ADDRESS = 0x41

5.3.3.3.6 Octotouch

GUI: Yes



Dependencies: -

Target system: Each OS (In same network as Octotouch-server)

Info:

You can set a parameters in file OctoTouch.py:

- devicename = "Octotouch"
- apikey = ""
- SAMPLERATE = 1

5.3.3.3.7 PIKO_Solarmodules

GUI: No

Dependencies: *pip3 install lxml*

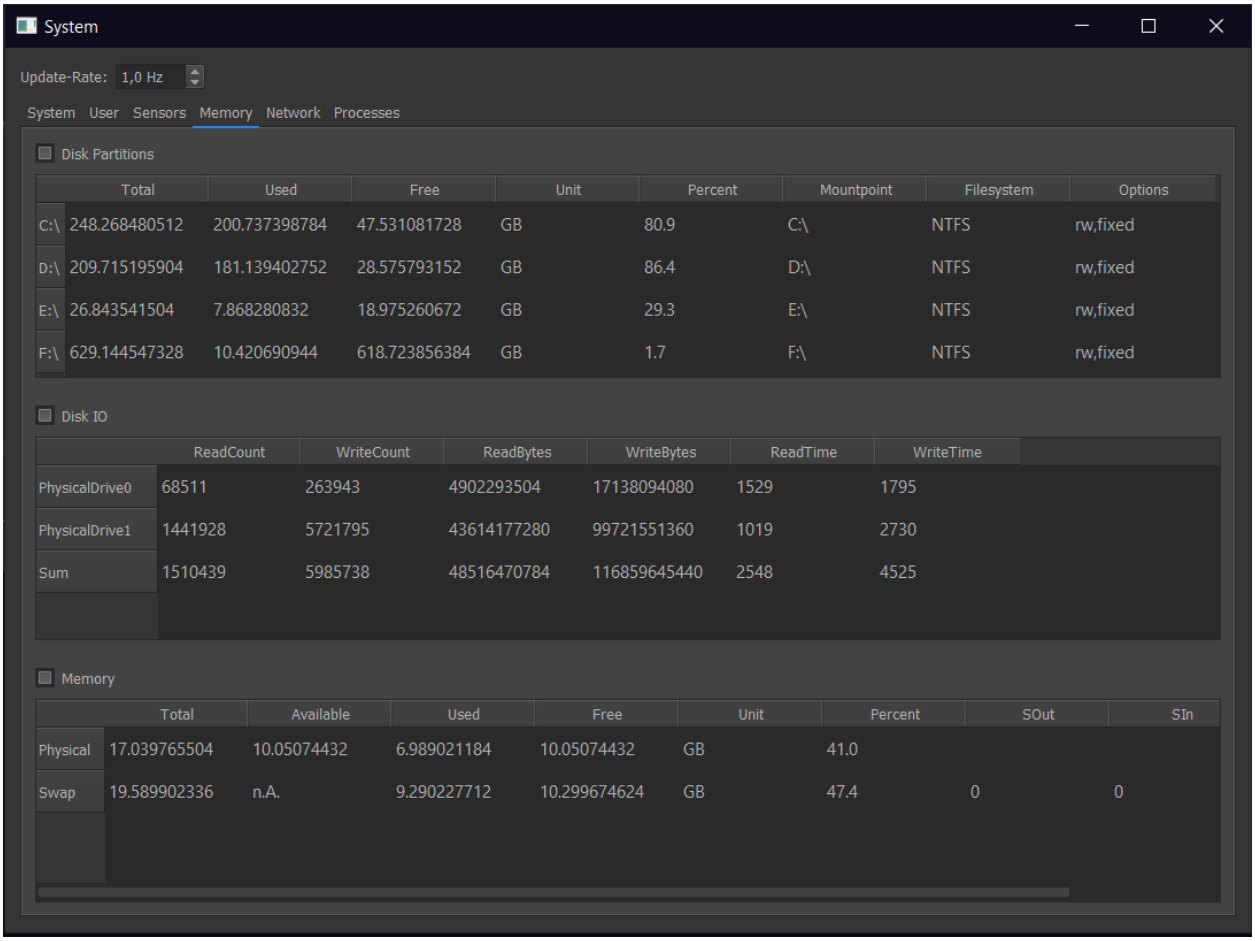
Target system: Each OS (In same network as PIKO modules)

Info:

- You can set a parameters in file INA219_Modul.py: - SAMPLERATE = 1/60# frequency in Hz (1/sec) - ADRESSES = ["IP1", "IP2", . . .] #You can specify multiple adresses

5.3.3.3.8 System

GUI: Yes



Dependencies: -

Target system: Each OS

Info:

5.3.3.3.9 ReflowOfen/ReflowPlatte

GUI: Yes

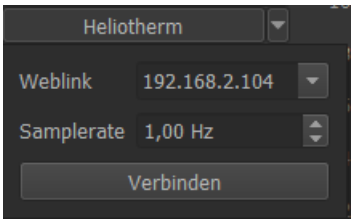
Dependencies: -

Target system: Each OS

Info:

5.3.3.3.10 Heliotherm

GUI: Yes



Dependencies: *pip3 install pyModbusTCP*

Target system: Each OS (In same network as Heliotherm heat pump)

Info:

5.3.3.3.11 Futtertrocknung

GUI: No

Dependencies: *pip3 install adafruit_CCS811 adafruit_DHT board busio*

Target system: RaspberryPi

Info:

5.4 Userdata

5.4.1 Document-Tree

User data is stored in the home directory of the current user

- home/USER/.RTOC/ on linux
- C:\user\USER\ .RTOC\ on windows

It contains the following files

```
.RTOC
├── backup
│   ├── localBackup1.json
│   ├── ...
├── devices
│   ├── Template
│   ├──   ├── ...
│   ├── ...
├── autorun_devices
├── config.json
├── globalActions.json
├── globalEvents.json
├── plotStyles.json
├── telegramActions.json
├── telegram_clients.json
```

5.4.2 backup (directory)

Used only, if backup is active and postgresql is not active. Currently this type of backup is not implemented.

5.4.3 devices (directory)

Place your plugins inside this folder! Each plugin must be in a folder, which should be named like the main-module of your plugin. For more information on how to make a plugin, click here: *Collecting data*

5.4.4 autorun_devices

Write plugin-names in each line of this file. These plugins will start with RTOC automatically (make sure the plugin-Thread is started in your __init__).

5.4.5 config.json

This file contains all RTOC configurations. Its separated in different topics:

5.4.5.1 global

Entry	Default	Type	Information
language	“en”	“en”,”de”	Selected language
recordLength	500000	int	Local recording limit
name	“RTOC-Remote”	str	Name displayed in Telegram
documentfolder	“~/.RTOC”	str	! Do not change !
webserver_port	8050	int	Port of webserver (disabled)
globalActionsActivated	False	bool	Global actions (in-)active
globalEventsActivated	False	bool	Global events (in-)active

5.4.5.2 postgresql

Entry	Default	Type	Information
active	False	bool	De/activate PostgreSQL database
user	“postgres”	str	PostgreSQL Username
password	“”	str	User’s password
host	“127.0.0.1”	str	Host of PostgreSQL-server
port	5432	int	PostgreSQL port
database	“postgres”	str	Name of PostgreSQL database
onlyPush	True	bool	Only push data automatically, if backup active

5.4.5.3 GUI

Entry	Default	Type	Information
darkmode	True	bool	De/activate darkmode (inactive)
scriptWidget	True	bool	Show/hide scriptWidget on startup
deviceWidget	True	bool	Show/hide deviceWidget on startup
deviceRAWWidget	True	bool	Show/hide deviceRAWWidget on startup
pluginsWidget	False	bool	Show/hide pluginsWidget on startup
eventWidget	True	bool	Show/hide eventWidget on startup
restoreWidgetPosition	False	bool	Save and restore window and widget positions
newSignalSymbols	True	bool	(inactive)
plotLabelsEnabled	True	bool	Show/hide signal-labels in graph
plotGridEnabled	True	bool	Show/hide grid in graph
showEvents	True	bool	Show/hide events in graph
grid":	[True, True, 1.0]	list	Grid configuration: [X-lines, Y-lines, linewidth]
plotLegendEnabled	False	bool	Show/hide legend in graph
blinkingIdentifier	False	bool	Show/hide blue blinking of signal-labels, when they are updated
plotRate	8	float	Updaterate of graph in Hz
plotInverted	False	bool	Invert plot (black-white/white-black)
xTimeBase	True	bool	Plot x-axis values as difference from actual timestamp
timeAxis	True	bool	Use time-ticks for x-axis
systemTray	False	bool	Dis/enable close to systemTray
signalInactivityTimeout	2	float	Time in seconds after Signal-Label turns yellow
autoShowGraph	False	bool	Automatically show new signals
antiAliasing	True	bool	Dis/Enable AntiAliasing
openGL	True	bool	Dis/Enable OpenGL
useWeave	True	bool	Dis/Enable Weave
csv_profiles	{}	dict	Allocation of imported signals is stored here

5.4.5.4 telegram

Entry	Default	Type Information	
active	False	bool	De/activate telegram-bot
token	""	str	Your telegram bot-token
default_eventlevel	0	0,1 or 2	Default eventlevel for new users
default_permission	'blocked'	'blocked','read', 'write' or 'admin'	Default user permissions for new users. First user is always admin!
inlineMenu	False	bool	Make the telegram menu inline or in KeyboardMarkup
onlyAdmin	False	bool	If True, only admins will be able to access the bot

5.4.5.5 tcp

Entry	Default	Type	Information
active	False	bool	De/activate TCP-server
port	5050	int	TCP-port
password	''	str	Optional password for TCP-encryption (AES)
knownHosts	{}	dict	Recent TCP-hosts for remote connection are stored here
remoteRefreshRate	1	float	Refresh-rate for remote session

5.4.5.6 backup

Entry	Default	Type	Information
active	False	bool	De/activate backup-thread
path	'~/.RTOC/backup'	str	Backup-directory (does not affect backup, if postgreSQL is active!)
clear	False	bool	Automatically clear local data after backup
autoIf-Full	True	bool	Automatically backup, if local recordLength is reached
autoOn-Close	True	bool	Automatically backup after closing RTOC
load-OnOpen	True	bool	Automatically load data after starting RTOC (if False, signals are still shown to make sure that IDs are allocated correctly)
intervall	240	float	Set backup-intervall in seconds
resam-ple	0	float	If != 0, signals will be resampled with given samplerate before creating backup

5.4.6 globalActions.json

This file contains all global actions. Read more about the event/action system here: *Event/Action system*

5.4.7 globalEvents.json

This file contains all global events. Read more about the event/action system here: *Event/Action system*

5.4.8 plotStyles.json

This file contains all signal styles, which are used by the GUI. Delete it, to reset all signal styles.

5.4.9 telegramActions.json

Use this file to add main-menu-entries in the telegram-bot. More information here: *Telegram Custom-menu*

5.4.10 telegram_clients.json

Information about telegram-clients is stored here: `clientID = {eventlevel = 0, shortcuts = [[],[]], first_name = "NAME", last_name = "NAME", permission = "admin", menu = "menu"}`

5.5 Telegram Communication

5.5.1 Telegram-Bot setup

5.5.1.1 Create a new Telegram-Bot

- Telegram must be installed on Smartphone/PC,...
- Open the following website: https://telegram.me/BotFather
- Click on 'SEND MESSAGE' (Telegram should open and create a new contact 'BotFather')
- Create a new bot with the command '/newbot
- Enter a name for the bot
- Enter a username for the bot
- You will receive a token which has to be entered in RTOC (looks like this: 123456789:AABBAABB66AA11_uGG22GH44AA55_3)

5.5.1.2 Configure RTOC for Telegram

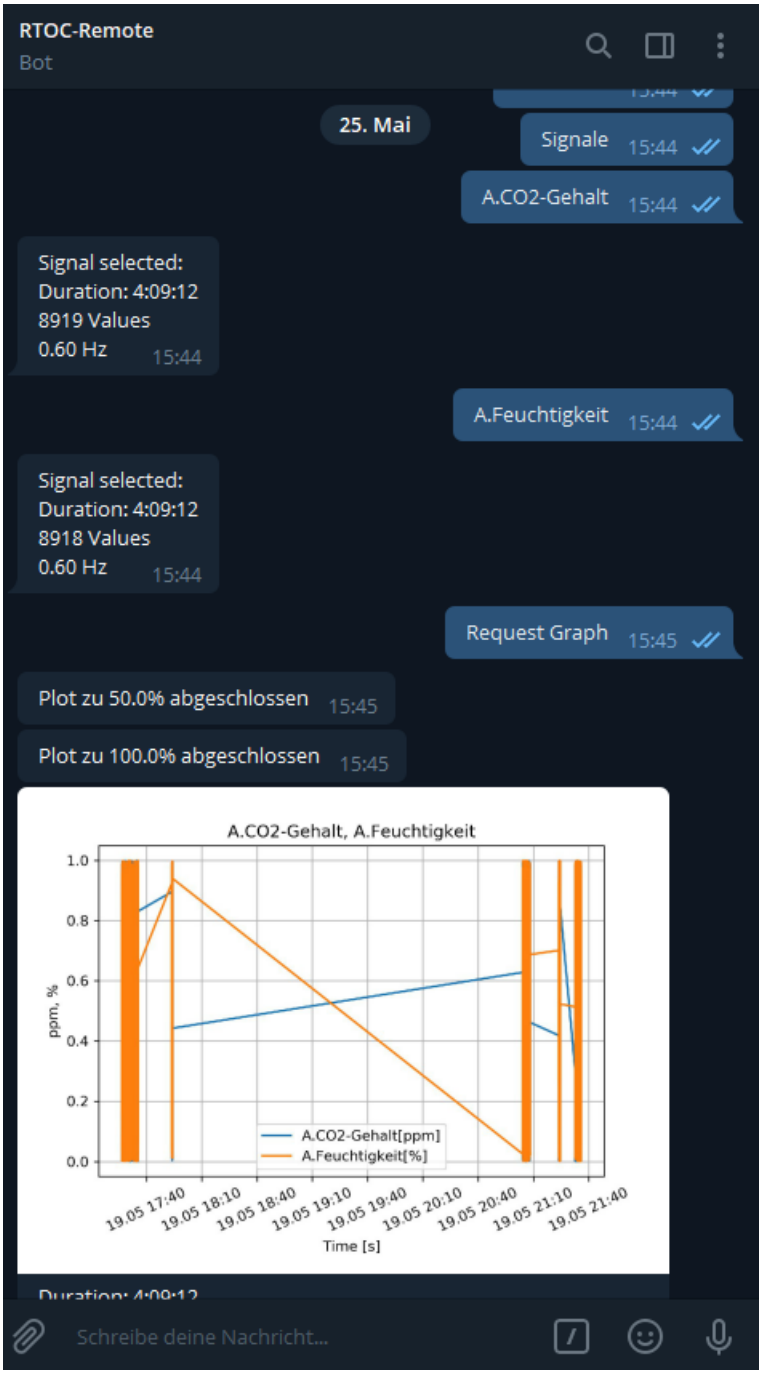
You can eather configure it in the settings of the RTOC GUI or edit the *config.json* file by yourself. Enter your Bot-Token there and activate it (`active=True`).

5.5.1.3 User permissions

By default any new client connected to this bot will have no permissions. You can change the default behavior in *config.json*.

The first client connected to the bot will automatically be admin. The admin has the permission to change the permissions of all users.

- **blocked**: No permissions at all
- **read**: Can receive event-notifications and plot signals
- **write**: Can create global Events/Actions, send Events/Signals, access devices
- **admin**: Full permissions. Can delete signals, access settings menu



5.5.2 Mainmenu

- <USER_ACTIONS>
- Latest values [**read**,**write**,**admin**]
- Signals [**read**,**write**,**admin**]
- Devices [**write**,**admin**]
- Send event/signal [**write**,**admin**]

- Automation [**write**,**admin**]
- Settings [**read**,**admin**, **write**] - Set Event Notification [**read**,**write**,**admin**] - General [**admin**, **write**] - Telegram-Bot [**admin**] - Backup [**admin**]

5.5.2.1 Latest values

Displays the most current measured value of each signal.

5.5.2.2 Signals

Contains a list of all signals. Clicking on a signal will select this signal. You can also select the time-period with `Select period`. `Generate plot` will send you a plot with your selected signals in the selected period of time.

`Show/Hide events` will show/hide events in the plot. Events are displayed as vertical lines with the event-text

`Delete signals` [**admin**] will delete all selected data (signals and time-period).

`Delete events` [**admin**] will delete all selected events (time-period).

5.5.2.3 Devices

Contains a list of all devices/plugins found on the RTOc server (see submenu: Device). Device - Start/stop device - Functions: List of all device functions. Execute by click - Parameters: List of all device parameters: Edit by click - Change samplerate - *Info: You can create main-menu-shortcuts for functions and parameters*

5.5.2.4 Send event/signal

Create a new event with Telegram. You can send measurements, which your doing manually, too.

5.5.2.5 Automation

Editor for *Event/Action system*.

5.5.2.6 Settings

5.5.2.6.1 Set Event Notification

Telegram-Notifications can be received from the RTOC-server if events occur. The notification level can be set here. (Averages, Warnings, Errors)

5.5.2.6.2 General

- Change recording length: Change the local recording length of the server.
- Change global samplerate: Change the samplerate of all plugins using `self.samplerate` or `self.setPerpetualTimer(func,samplerate)`.
- TCP-Server: On/Off: En/disable TCP-server
- Restart server: Restart host computer

5.5.2.6.3 TelegramBot

- Telegram clients: List of connected clients. Admins can change user-permissions.

5.5.2.6.4 Backup-Settings

- Configure all backup options.
- Delete signals: Deletes all signals and events.
- Resample database

5.5.3 Telegram Custom-menu

The file *telegramActions.json* contains dicts with actions, that will be shown in the main menu and can be executed by any user. If the action-name (key) starts with '_' only admins will be able to see this button.

Here is an example to send a screenshot

```
{
       "Screenshot": """

    import pyscreenshot as ImageGrab
    dir = self.config['global']['documentfolder']
    im = ImageGrab.grab()\nim.save(dir+'/telegram_overview.png')
    return 'picture', dir+'/telegram_overview.png'
    """

}
```

A telegram action must return either a text, a picture or any other file.

`return 'text', 'My example text'` to return a text message.

`return 'picture', <dir/to/picture.jpg>` to return a picture.

`return 'document', <dir/to/file>` to return any other file.

5.6 Controlling and automation

5.6.1 Overview

Own scripts can be used during the runtime:

- modify the measurement data
- Adjusting plugin parameters and calling functions
- Create new signals
- manual and automatic controlling, ...

In general, you write a Python script like any other, you can import libraries, etc. However, you should pay attention to performance.

RTOC provides an own library for scripts: `RTLogger.scriptLibrary`

There are three places, where custom scripting is possible.

- *Run scripts in GUI*
- Telegram-menu-entries: *Telegram Communication*
- *Event/Action system*

5.6.2 Available functions and libraries

5.6.2.1 Python libraries

There are several python libraries automatically imported in scripts

```
import math
import numpy as np
import sys
import scipy as sp
from .RTLogger import scriptLibrary as rtoc
```

5.6.2.2 Functions to interact with RTOC

5.6.2.3 Access to plugin parameters and signals in scripts

All signals can be references in scripts in these ways

```
[x],[y] = Device.Signalname
[x] = Device.Signalname.x
[y] = Device.Signalname.y
c = Device.Signalname.latest
```

plugin parameters and functions can be accessed in this way

```
Device.FUNCTION(...)
Device.PARAMETER = ...
```

5.6.2.4 Special stuff

The actual timestamp is available in the global variable `clock`. You can use the default `print()` function for text-output in console, GUI and telegram.

You can define variables **global**. This ensures, this variable will remain until the next call of this script. Example

```
global VARNAME = 0
```

If-else conditions aren't always suitable for real-time operations. You cannot trigger rising/falling for example. Therefore you can use a trigger to call the code inside a condition only once

```
trig CONDITION:
  print('Hello')
```

This example will only print 'Hello', if CONDITION changes from `False` to `True`.

5.6.2.5 RTOC library

5.6.3 Event/Action system

The Event/Action System allows Python code to be executed when events occur. These pieces of code have the same possibilities as scripts.

5.6.3.1 Global actions

Global actions are stored in the file *globalActions.json*.

Example

```
{
  "action2": {
    "listenID": ["testID"],
    "parameters": "",
    "script": "Generator.gen_level = 1",
    "active": False

  }

}
```

This JSON-file contains dicts - each describing one action.

Parameter	Datatype	Definition
`listenID`	`list`	The event-IDs of events, which will trigger this action
`parameters`	`str`	Unused
`script`	`str`	The script, which will be executed
`active`	`bool`	If True, this action will be active

5.6.3.2 Global events

Global events are stored in the file *globalEvents.json* unlike events created in plugins.

Example

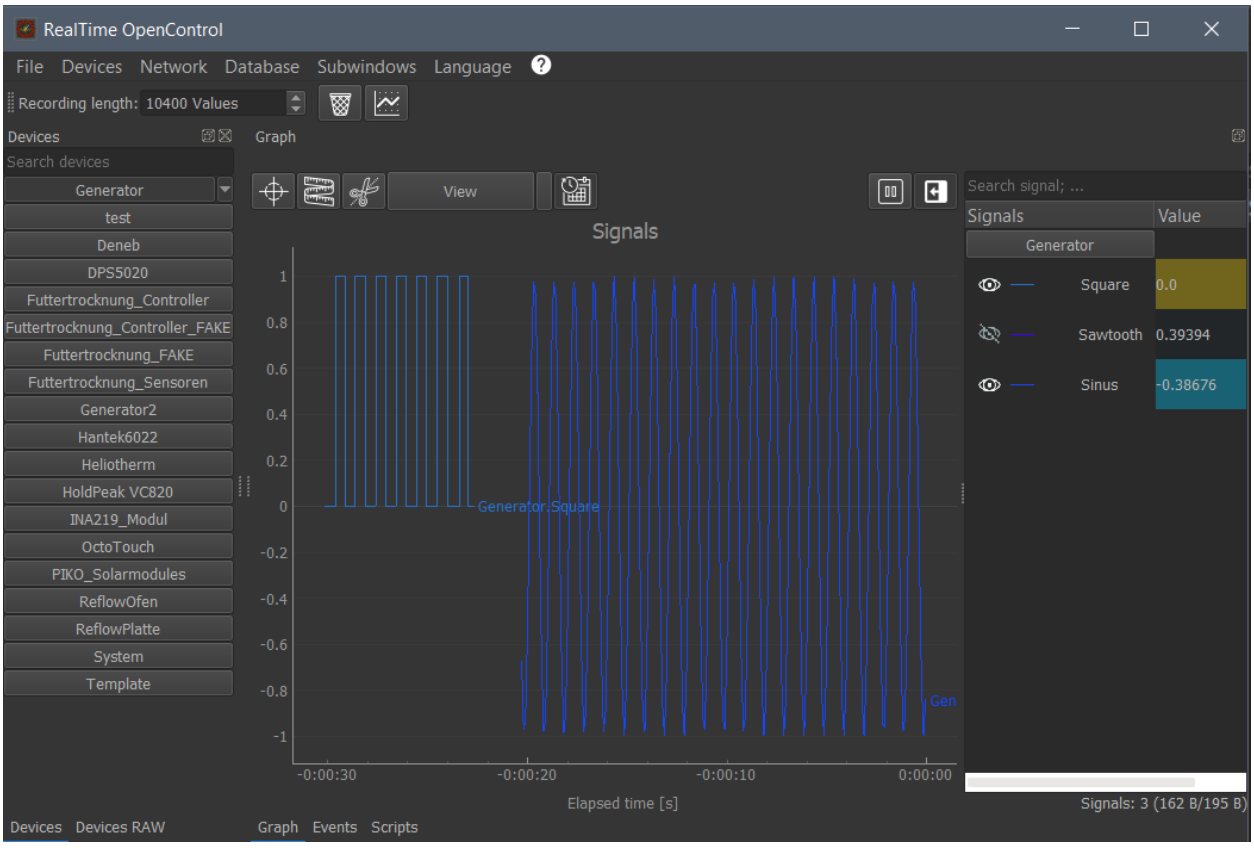
```
{
  "myevent": {
    "cond": "Generator.Square.latest >= 2",
    "text": "It is big",
    "return": " ",
    "priority": 0,
    "id": "testID",
    'trigger': 'rising',
    'sname': '',
    'dname': '',
    'active': True

  }

}
```

This JSON-file contains dicts - each describing one event.

Parameter	Datatype	Definition
`cond`	`str`	The condition triggering the event.
`trigger`	`str`	Change trigger-mode. `rising`, `falling`, `both`,"true", `false`. If `rising`, the event will be triggered if cond changes from `False` to `True`. If `falling`, the other way around. `both` for rising and falling. `true` for always, if condition is true. `false` for always, if condition is false.
`active`	`bool`	If True, this event will be active.
`id`	`str`	The event-ID used to trigger actions.
`return`	`str`	Unused
`text`	`str`	See `LoggerPlugin.event()`
`dname`	`str`	...
`sname`	`str`	
`priority`	`0, 1 or 2`	

Global actions and events can also be configured during runtime in the telegram-bot.

5.7 Graphical user interface



You can use the GUI to control your plugins, manipulate your measurements, try scripts, import data and configure your settings. You can also view and control RTOC-servers remotely. Unfortunately this is limited to short recordings due to the TCP-protocol, which cannot handle long datasizes. This will maybe be fixed in future releases.

5.7.1 Titlebar

Systembar-structure

```
Systembar
├── File
│   ├── Load session
│   ├── Save session
│   ├── Import data
│   ├── Export data
│   ├── Minimize to system tray
│   ├── Settings
│   ├── Quit
├── Devices
│   ├── Add/Remove devices
├── Network
│   ├── TCP-Server
│   │   ├── Active
│   │   ├── Password-Protection
│   │   ├── Port: 5050
│   ├── Telegram-Bot
│   │   ├── Active
│   │   ├── Bot-Token
│   ├── Connect to remote host
│   │   ├── New host
│   │   ├── ...
│   ├── Active connections
│   │   ├── ...
│   ├── Update-Rate: 1Hz
│   ├── Search RTOC-Server
├── Database
│   ├── Save local data in database
│   ├── Reset local data with data from database
│   ├── Export database
├── Subwindows
│   ├── Devices
│   ├── Plugins
│   ├── Script
│   ├── Events
│   ├── DevicesRAW
├── Language
│   ├── German
│   ├── English
├── Help
│   ├── Clear cache
│   ├── RTOC Help
│   ├── Plugin-repository
│   ├── Check for updates
│   ├── About
```

Subtitlebar

- Number-Edit for local recordLength

- Delete data: Deletes all existing data.
- Open new plot window. You can simply drag'n'drop signals from one window into another.

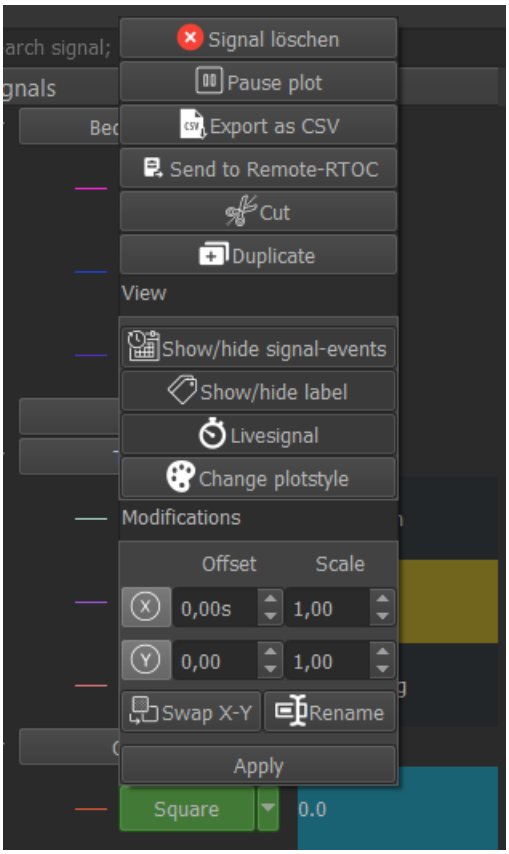
5.7.2 Device Widget

The device widget (left) holds all plugins. Each plugin is represented by a button. You can start/stop plugins with these buttons. If the plugin has a GUI and `smallGUI` is True, the gui will be available as a button-dropdown.

5.7.3 Signal Widget

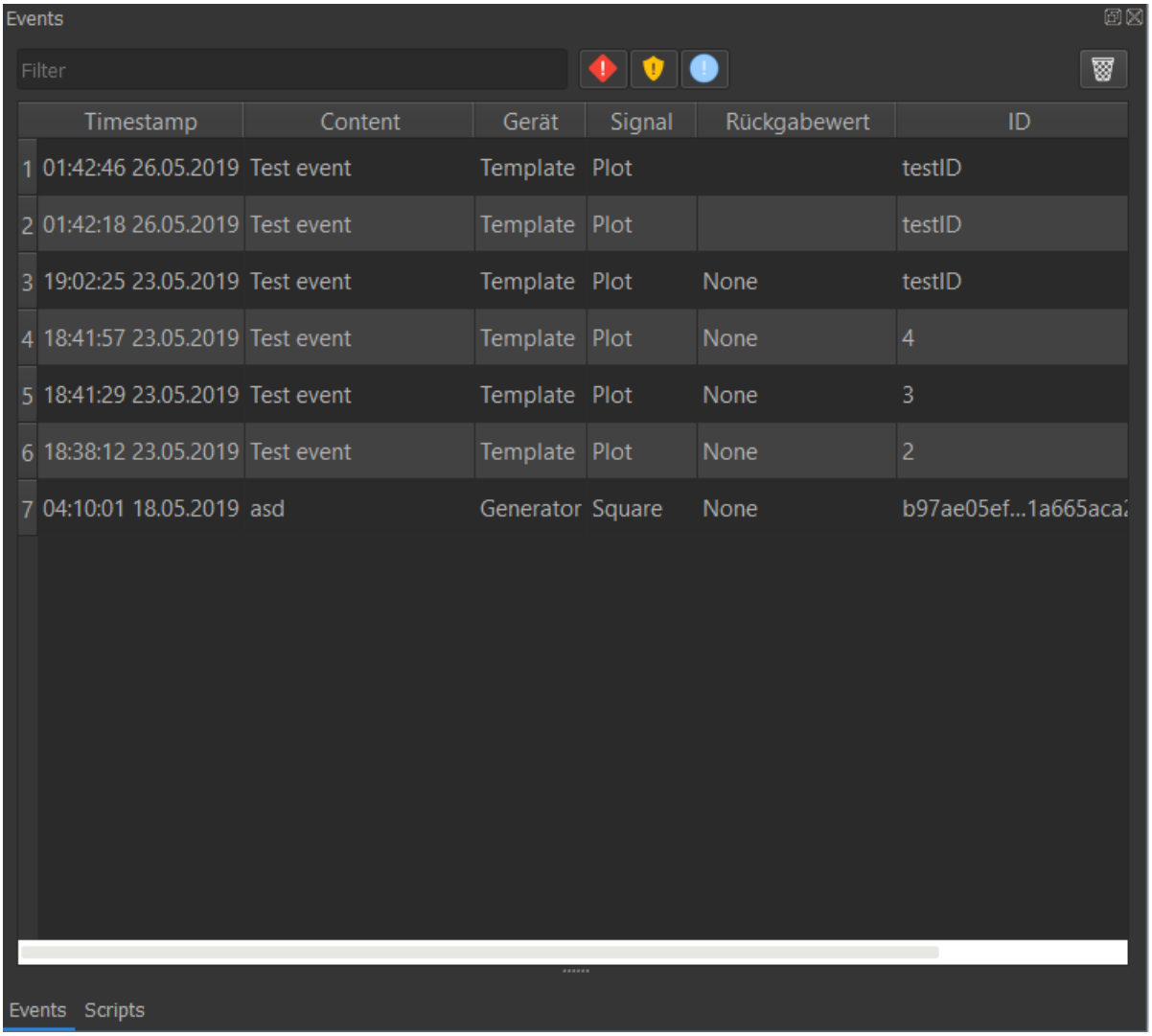
The signal widget (right) hold all signals. You can toggle each signal by clicking it (red=hidden, green=visible). Next to the button is the latest value.

Each signal also has a dropdown-menu



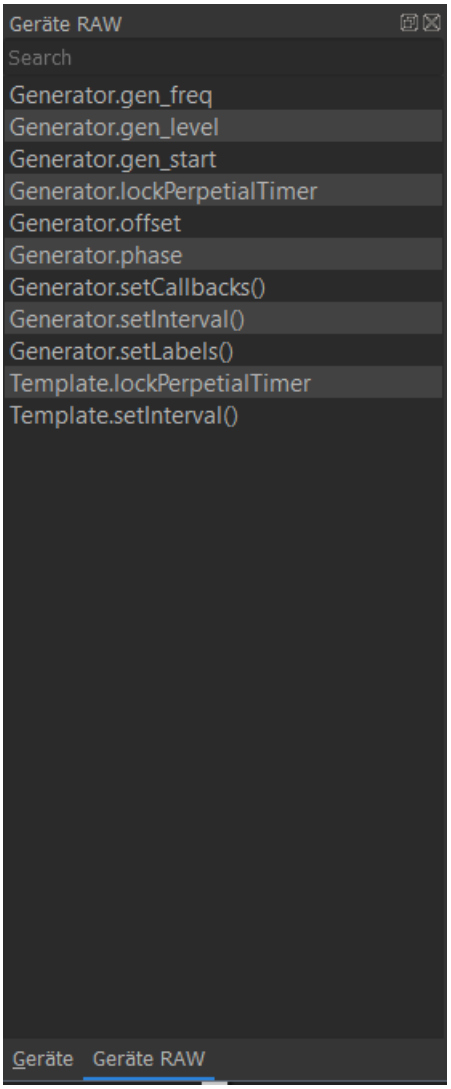
Check out each menu-entry by yourself, please.

5.7.4 Event Widget



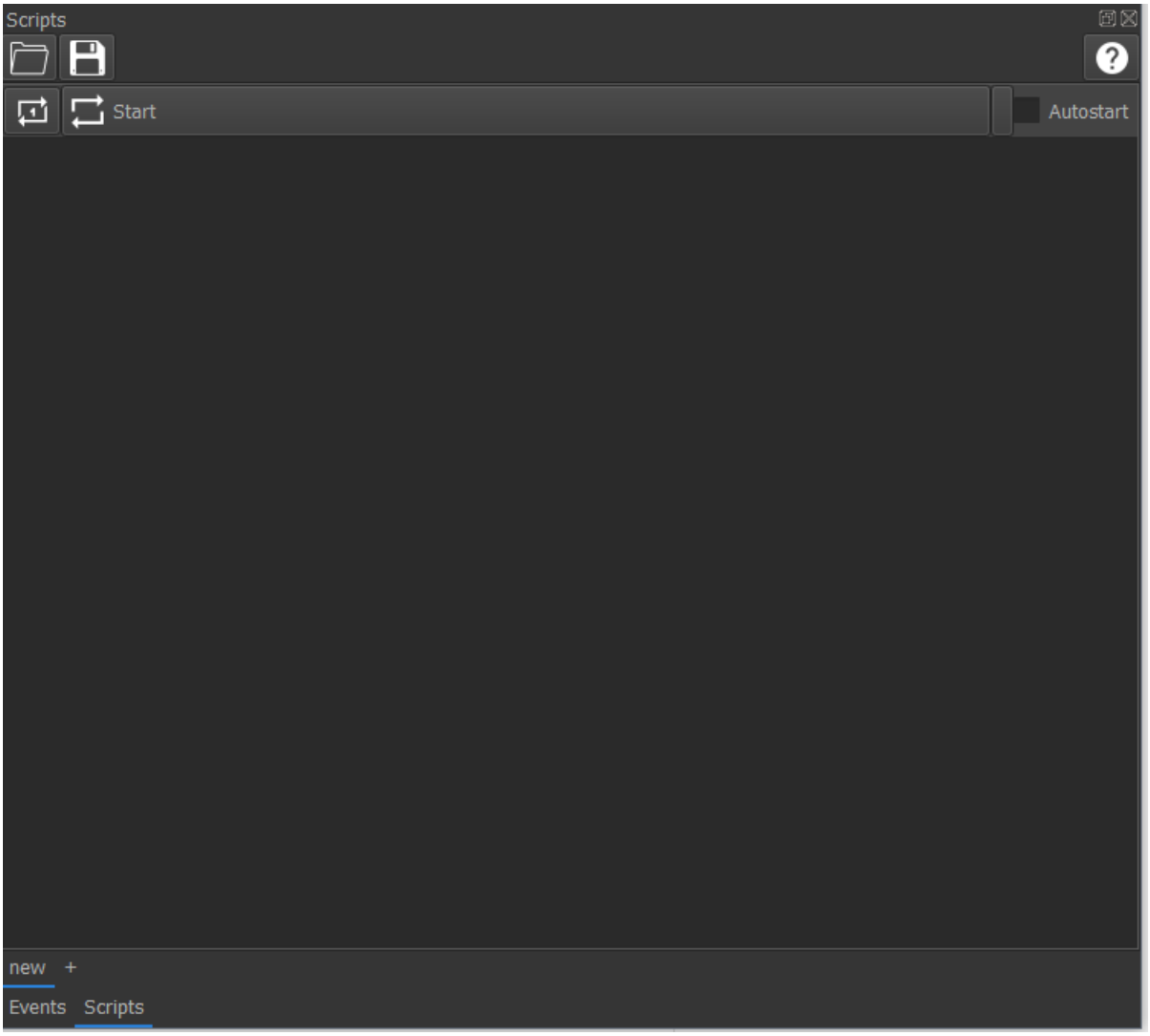
The event widget shows a list with all events. You can filter the events by text or hide specific priorities. On the right top is a button to delete all events.

5.7.5 DevicesRAW Widget



This widget holds a list with all plugin functions and parameters

5.7.6 Script Widget



This widget is a simple text-editor for scripts. If you want to learn more about scripts, check out *Controlling and automation*.

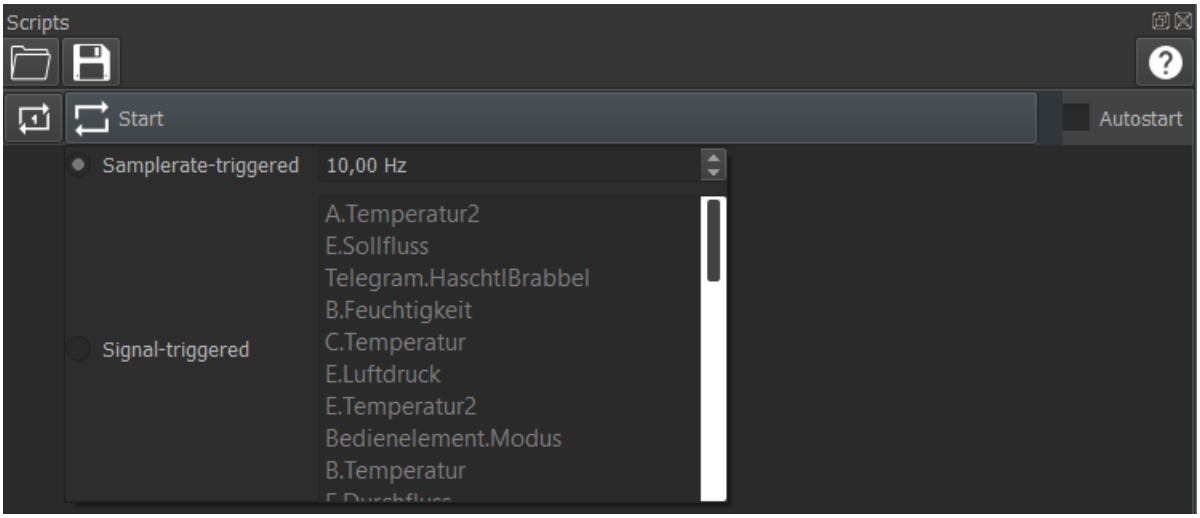
5.7.7 Run scripts in GUI

Click on one of the buttons at the top of this widget to run the code either once or repeated.

You can also load/save scripts from/to file.

At the right top is a help window with lists for all signals, parameters and functions.

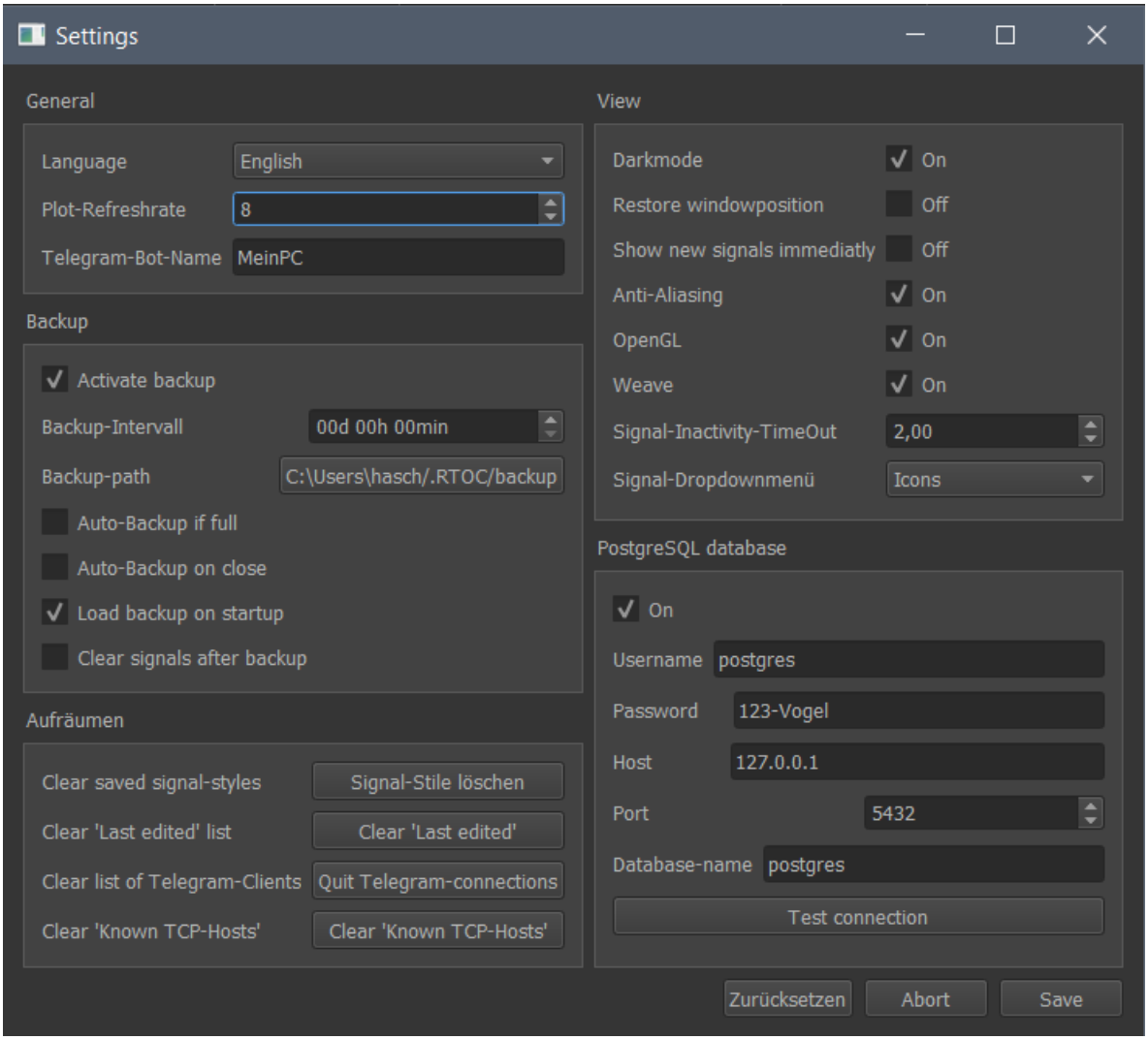
5.7.7.1 Trigger-System



Scripts are executed in two different ways (can be selected from the "Start" button's drop-down menu in the ScriptWidget):

- Samplerate-Triggered: script is executed periodically
- Signal-Triggered: Script is executed, if a new signaldata is received. You can select multiple trigger-signals. In this case, the latest xy-pairs of the triggered signals can still be modified.

5.7.8 Settings Widget



This widget can modify the *config.json*

5.7.9 Import/Export signals/sessions

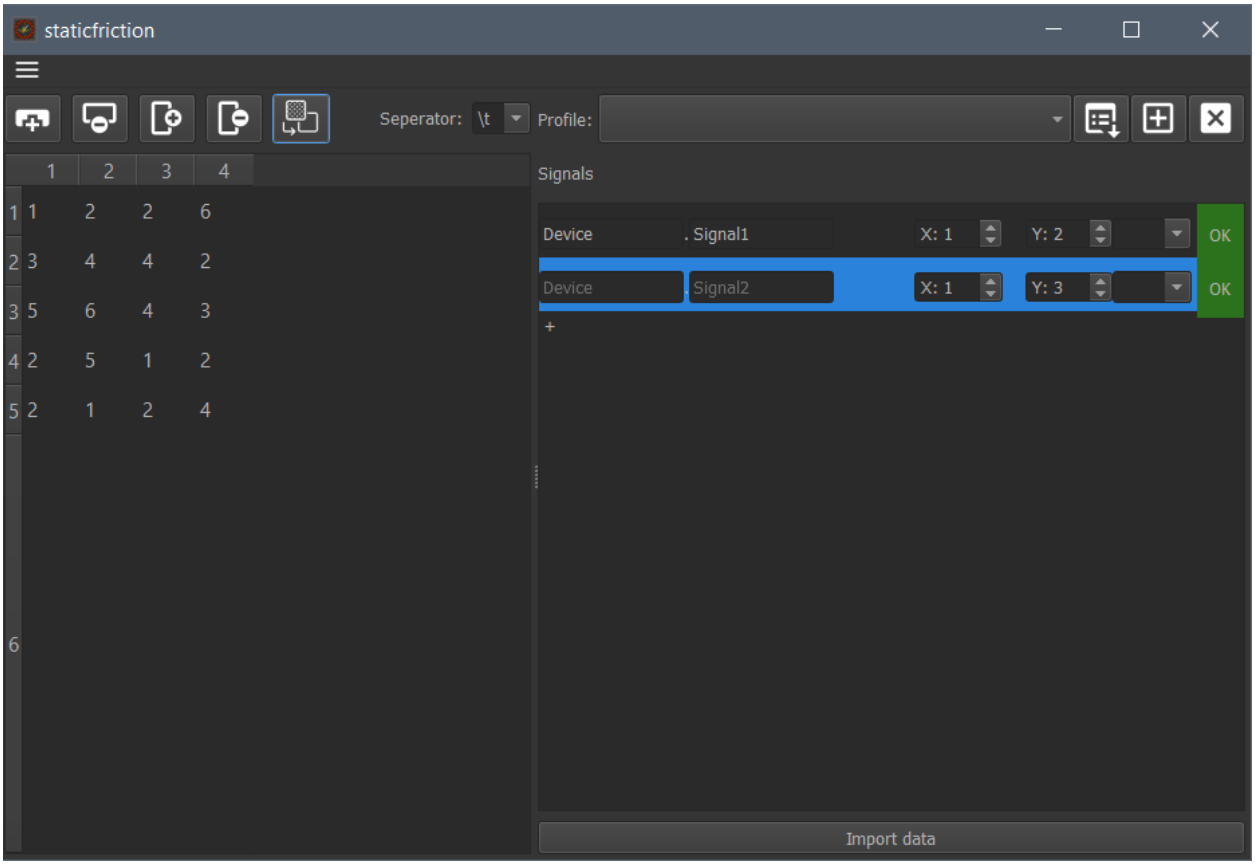
5.7.9.1 Import session

1. Open “File”->”Load session” in the menubar
2. Select a file you want to import

or

1. Drag’n’Drop a file or copied data into RTOC

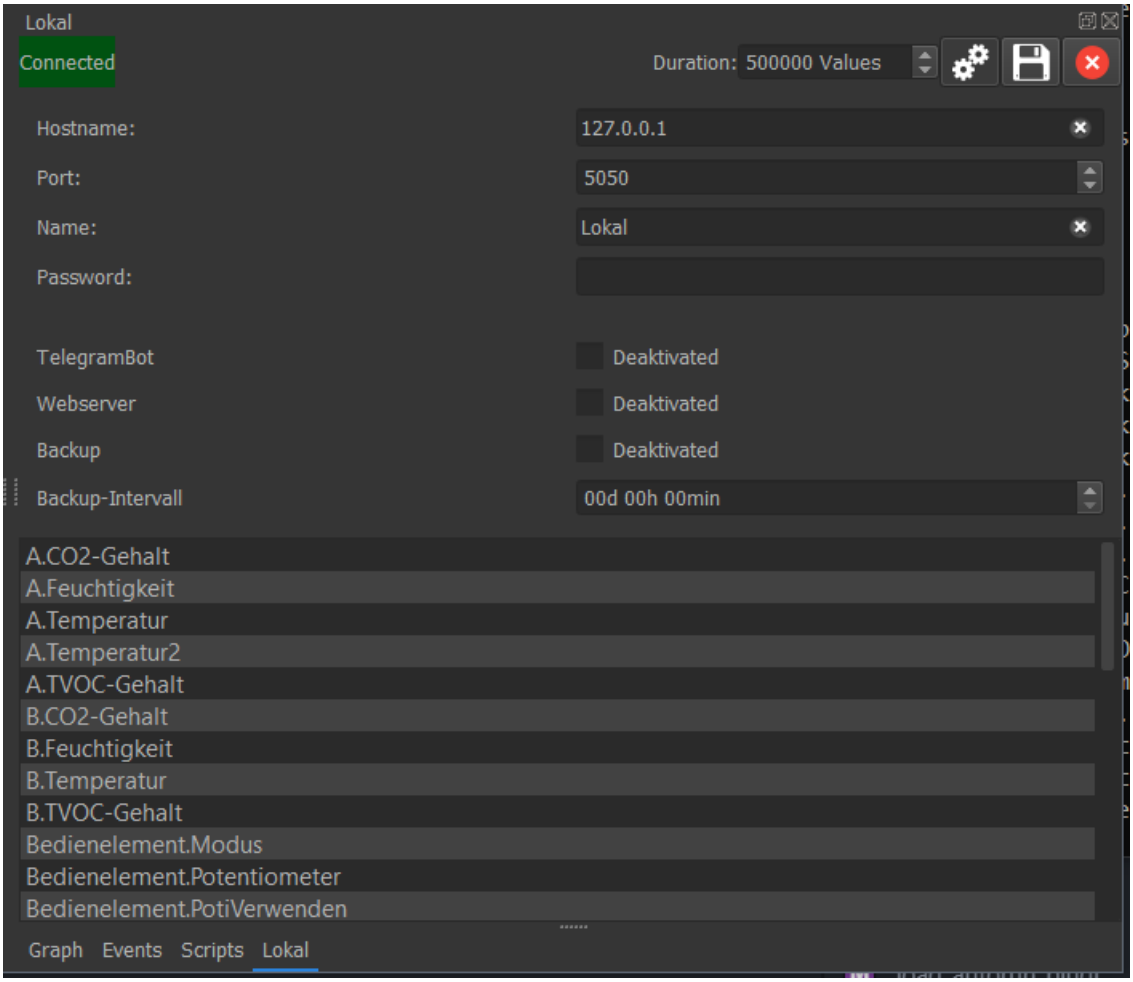
5.7.9.2 Import XLSX, MATLAB, CSV



On the left side is the data-table. You can modify it to your needs.

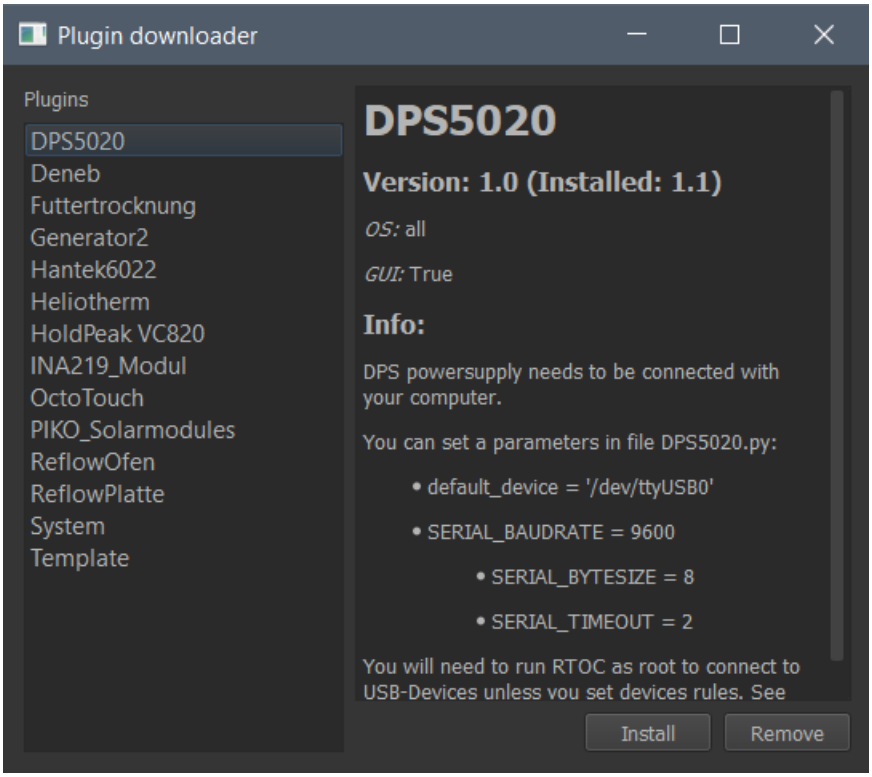
On the right side you can define signals, that will be taken from the data-table. 1. Click (+) to add a new signal. 2. Set a signal and devicename (not needed) 3. Select columns for X and Y data. If X-Column is 0, X-data will be generated automatically 4. The color on the right of each signal indicates, if this signal can be imported (mouse-over gives more information on failure) 5. Click “Import data” to load the signals to RTOC. Invalid signals will be skipped

5.7.10 Remote-control via TCP



You can connect to any remote RTOC in the 'Network'-menu of the *Titlebar*.

5.7.11 Plugin-Downloader



This tool can automatically download, update and remove signals from the *Plugin repository*.

5.8 Run webserver

5.8.1 RTOC Webserver

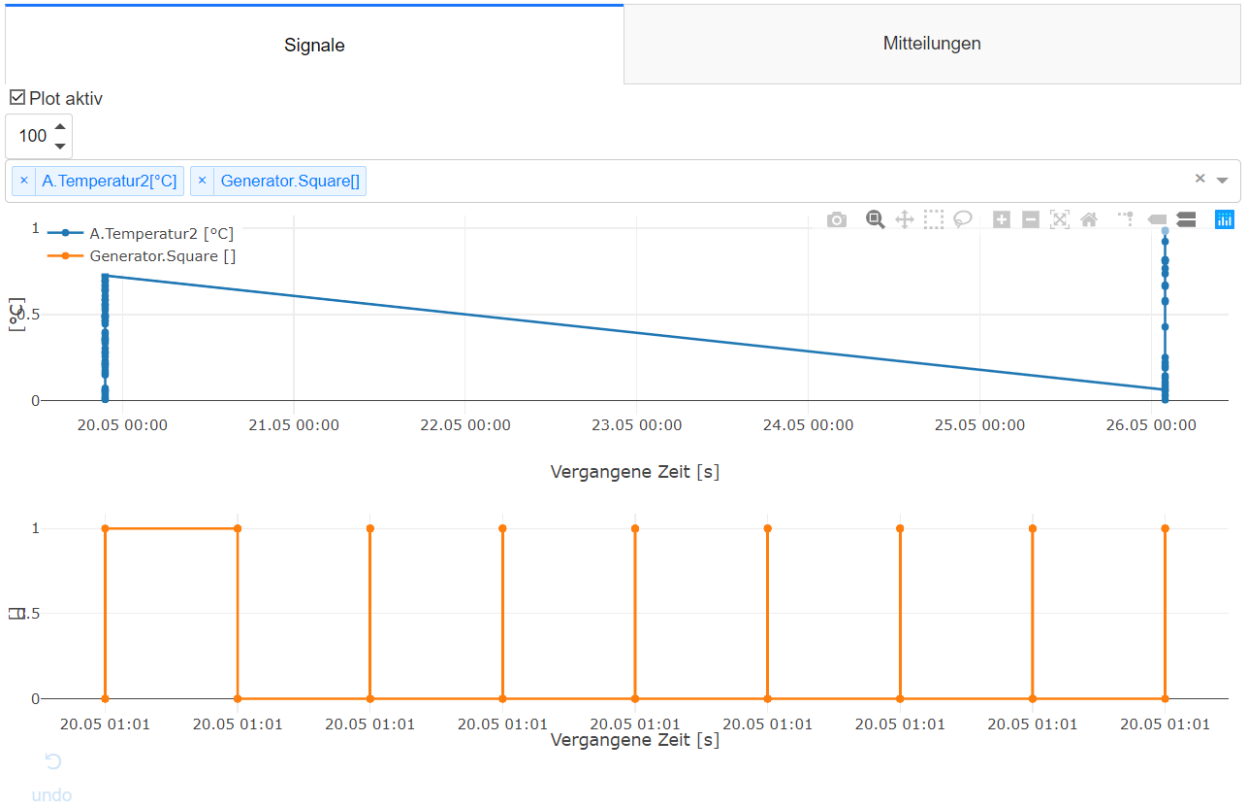
Since *v2.0* the webserver must be started with `python3 -m RTOC.RTLogger -w`.

If you have enabled the postgreSQL database, the webserver will run standalone and read the data from the database only.

If you do not have enabled postgreSQL, RTOC will start with the webserver. Therefore, it's **not** possible to run GUI and webserver at the same time (without postgreSQL) !

The webserver is by default reachable on port 8050. You can view the webpage from the same device with *http://localhost:8050* in any browser. If you want to access the webpage from any other network device, you must know the IP-address or hostname. For a RaspberryPi for example *http://raspberrypi:8050*.

5.8.1.1 Plots



Right beneath the tab-bar is the signal-selection field. You can select multiple signals, which will be plotted afterwards. Depending on the size of the signals, this can take some time. You can also plot only the latest X values by editing the number-edit left to the signal-selection field.

The webserver will automatically generate multiple figures for each unit.

5.8.1.2 Events

Signale				Mitteilungen		
Zeitpunkt	Typ	Gerät	Signal	Inhalt	ID	Rückgabewert
filter data...						
2019-05-18 04:10:01:662610	Information	Generator	Square	asd	None	b97ae05efa66871a269dcb97c0575291a665aca2
2019-05-23 18:38:12:467260	Information	Template	Plot	Test event	None	2
2019-05-23 18:41:29:159223	Information	Template	Plot	Test event	None	3
2019-05-23 18:41:57:936598	Information	Template	Plot	Test event	None	4
2019-05-23 19:02:25:986256	Information	Template	Plot	Test event	None	testID

undo

This tab shows a simple list of all events

5.9 TCP Communication

TCP Port: 5050 (by default)

The content is formatted as a JSON file (Python: dict) with the following keys (all optional). For more information: `RTLogger.NetworkFunctions`

dict-key	Description
`plot = False`	`True`: x and y data are plotted as one signal. `False`: x and y data are signals in pairs. See `LoggerPlugin.plot()` for more information.
`x = [0,1,2,3,4,5]`	X data to be sent. If `x` is not set and `plot = False`, `time.time()` is set as x-value. If `plot = True`, the indices of the `y` data are used. See `LoggerPlugin.plot()` for more information.
`y = [1,2,3,4,5,6]`	Y-data to be sent. See `LoggerPlugin.plot()` for more information.
`sname = ["signalname"]`	List of signal names with `plot = False`, only one element with `plot = True`. See `LoggerPlugin.plot()` for more information.
`dname = "devicename"`	Device name to be transmitted. See `LoggerPlugin.plot()` for more information.
`unit = "unit"`	signal unit. See `LoggerPlugin.plot()` for more information.
`event = {text = "", dname='', sname='', x=clock, priority=0}`	Create an event. See `LoggerPlugin.event()` for more information.
`getLatest= True`	If `getLatest=True`, RTOC returns a Dict of the most current measured values. The signal names are the keys. See `NetworkFunctions.getLatest()` for more information.
`getSignalList = True`	If `getSignalList=True`, RTOC returns a list of signal names. See `NetworkFunctions.getSignalList()` for more information.
`getEventList = True`	If `getEventList=True`, RTOC returns a list of all events. See `NetworkFunctions.getEventList()` for more information.
`getPluginList = True`	If `getPluginList =True`, RTOC returns a Dict of the plugins containing the plugin functions, parameters and the status of the plugin. See `NetworkFunctions.getPluginList()` for more information.
`getEvent = ['Device.Signal',...]`	Server request for the events of a signal. See `NetworkFunctions.getEvent()` for more information.
`getSignal = ['Device.Signal',...]`	Server request for signal data. See `NetworkFunctions.getSignal()` for more information.
`plugin = {...}`	TCP access to plugins with Dict. See `NetworkFunctions.handleTcpPlugins()` for more information.
`logger = {...}`	RTOC default functions. See `NetworkFunctions.handleTcpLogger()` for more information.

As response RTOC delivers a dict with the following keys:

dict-key	Description
error = False	If True, an error has occurred in the transmission
sent = False	Is True if data (x,y) has been transmitted to the server.
signalList = []	Contains list of devices, at getSignalList-Request
pluginList= {}	Dict with plugins, with getPluginList-Request
signals = {}	Dict with signals, with getSignal-Request
events = {}	Dict with events, at getEvent-Request
latest = {}	Dict with latest measured values, at getLatest-Request

5.9.1 Python example (just with `jsonsocket`)

This example uses the module `jsonsocket`:

```
import jsonsocket

data = {'x':[0,1,2,3],'y':[1,2,3,4],'dname':'Test','sname':['T1','T2','T3','T4']}
sock = jsonsocket.Client()
sock.connect('127.0.0.1', 5050)
sock.send(data)
response = self.sock.recv()
self.sock.close()

print(response)
# {'error':False, 'sent':True}
```

5.9.2 Python example with `LoggerPlugin`

The following functions simplify access via TCP and are included in `LoggerPlugin`.

This is strongly recommended for your project!:

```
import RTOC.LoggerPlugin as LoggerPlugin

class RtocClient(LoggerPlugin):
    def __init__(self, address="localhost", password=None):
      super(Plugin, self).__init__(None, None, None)
      self.setDeviceName('MyExampleTCPClient')

      self.createTCPClient(address="localhost", password=None, tcpport=5050,␣
↪threaded=False)

      response = self.sendTCP(x=[0,1,2,3],y=[1,2,3,4],dname='Test',sname=['T1','T2',
↪'T3','T4']

      print(response)
      # {'error':False, 'sent':True}
```

5.10 Backend Source-code

5.10.1 LoggerPlugin.py

5.10.2 jsonsocket.py

class Client

Bases: `object`

A JSON socket client used to communicate with a JSON socket server. All the data is serialized in JSON. How to use it:

Parameters **keyword** (*str or None*) – Set a keyword for encrypted communication. Leaf blank for unsecure connection. (default: None)

close()

Close the client-socket. Do this at the end of every single communication!

connect(*host*, *port*, *keyword=None*, *reuse_port=True*, *timeout=5*)

Establish a connection to a host (server)

Parameters

- **host** (*str*) – Hostname (e.g. 0.0.0.0)
- **port** (*int*) – Port to bind tcp-socket (e.g. 5050)
- **keyword** (*str or None*) – Set a keyword for encrypted communication. Leaf blank for unsecure connection. (default: None)
- **reuse_port** (*bool*) – Enable/disable reuse_port (default: True)

recv()
Receives a dict from server

Returns The dict sent by the connected server

Return type data (dict)

recv_and_close()
Receives a dict from server and closes connection.

Returns The dict sent by the connected server

Return type data (dict)

send(*data*)
Send a dict to the connected server

Parameters **data** (*dict*) – The dict you want to transmit.

setKeyword(*keyword=None*)
Set a keyword to protect the tcp communication.

Parameters **keyword** (*str or None*) – A save passcode or None to disable protection

exception NoPasswordProtectionError(*expression*, *message*)
Bases: exceptions.Exception

Raised when a password was provided, but server has no protectionAttributes.

expression -- input expression in which the error occurred

message -- explanation of the error

exception PasswordProtectedError(*expression*, *message*)
Bases: exceptions.Exception

Raised when the server may be password protected.

expression -- input expression in which the error occurred

message -- explanation of the error

class Server(*host*, *port*, *keyword=None*, *reuse_port=True*, *timeout=5*)
Bases: object

A JSON socket server used to communicate with a JSON socket client. All the data is serialized in JSON.

Parameters

- **host** (*str*) – Hostname (e.g. 0.0.0.0)
- **port** (*int*) – Port to bind tcp-socket (e.g. 5050)
- **keyword** (*str or None*) – Set a keyword for encrypted communication. Leaf blank for unsecure connection. Note: This will not encrypt host-intern-communication. (default: None)

- **reuse_port** (*bool*) – Enable/disable reuse_port (default: True)

accept()
Accept a new client connection

Returns client

close()
Close the server-socket. Do this at the very end of your communication!

recv()
Receives a dict from client

Returns The dict sent by the connected client

Return type data (dict)

send(*data*)
Send a dict to the connected client

Parameters **data** (*dict*) – The dict you want to transmit.

setKeyword(*keyword=None*)
Set a keyword to protect the tcp communication.

Parameters **keyword** (*str or None*) – A save passcode or None to disable protection

exception WrongPasswordError(*expression*, *message*)
Bases: exceptions.Exception

Raised when the password is wrong.

expression -- input expression in which the error occurred

message -- explanation of the error

deserializeJSON(*json_bytes*)

pad(*text*, *padding=16*)

readBlock(*socket*, *blockLength*)

5.10.3 RTOC.RTLogger Submodules

5.10.3.1 RTOC.RTLogger.Daemon module

Generic linux daemon base class for python 3.x.

Source: https://web.archive.org/web/20160320091458/http://www.jejik.com/files/examples/daemon3x.py

class Daemon(*pidfile*)
A generic daemon class.

Usage: subclass the daemon class and override the run() method.

daemonize()
Deamonize class. UNIX double fork mechanism.

delpid()

restart()
Restart the daemon.

run()
You should override this method when you subclass Daemon.

It will be called after the process has been daemonized by start() or restart().

start()
Start the daemon.

stop()
Stop the daemon.

5.10.3.2 RTOC.RTLogger.DeviceFunctions module

5.10.3.3 RTOC.RTLogger.EventActionFunctions module

class EventActionFunctions
This class contains all global event/action-specific functions of RTLogger

addGlobalAction(*name='testAction'*, *listenID='testID'*, *script=''*, *parameters=''*, *active=True*)

addGlobalEvent(*name='testEvent'*, *cond=''*, *text='TestEvent'*, *priority=0*, *retur=''*, *id='testID'*, *trigger='rising'*, *sname=''*, *dname=''*, *active=True*)

checkGlobalActions(*actions*)
Checks if actions are valid. Will add the key 'errors' to each event. 'errors' will be True, if it's not valid.

Parameters **actions** (*dict*) – actions to be checked.

Returns Input-actions + 'errors'-key

Return type actions (dict)

checkGlobalEvents(*events*)
Checks if events are valid. Will add the key 'errors' to each event. 'errors' will be True, if it's not valid.

Parameters **events** (*dict*) – events to be checked.

Returns Input-events + 'errors'-key

Return type events (dict)

loadGlobalActions()
Loads global actions from file and stores them in dict 'self.globalActions'

loadGlobalEvents()
Loads global events from file and stores them in dict 'self.globalEvents'

performGlobalActions(*id*, *value*)

performGlobalEvents(*y*, *unit*, *devicename*, *signalname*, *x=None*)
Performs global events, if their conditions are fullfilled. This function is called any time data is added to signals.

Parameters

- **y** (*float*) – y-value of signal (not needed)
- **unit** (*str*) – unit of signal (not needed)
- **devicename** (*str*) – devicename of signal
- **signalname** (*str*) – signalname of signal
- **x** (*float*) – x-value of signal (not needed)

printGlobalActions()

printGlobalEvents(*wo=True*)

removeGlobalAction(*key*)

removeGlobalEvent(*key*)

saveGlobalActions()
Saves global actions from 'self.globalActions' to file.

saveGlobalEvents()
Saves global events from 'self.globalEvents' to file.

triggerGlobalAction(*key*)

triggerGlobalEvent(*key*)

translate(*id*, *text*)

5.10.3.4 RTOC.RTLogger.NetworkFunctions module

5.10.3.5 RTOC.RTLogger.RTLogger module

5.10.3.6 RTOC.RTLogger.RTOC_Web module

5.10.3.7 RTOC.RTLogger.RTOC_Web_standalone module

5.10.3.8 RTOC.RTLogger.RTRemote module

5.10.3.9 RTOC.RTLogger.RT_data module

5.10.3.10 RTOC.RTLogger.ScriptFunctions module

class ScriptFunctions
This class contains all script-execution-specific functions of RTLogger

checkCondition(*conditionStr*)

createConditionFunction(*s*)

createFunction(*s*)

executeScript(*scriptStr*)

generateCode(*s*, *condition=False*)

generateTriggerCode(*scriptStr*)

printfunction()

replaceGlobalVariables(*s*)

replaceLibraryFunctions(*s*)

replaceLoggerFunctions(*s*)

replacePluginMethods(*s*)

replacePluginParameters(*s*)

replaceSignalNames(*s*)

triggerExpressionHandler(*expression*)

5.10.3.11 RTOC.RTLogger.importCode module

importCode(*code*, *name*, *add_to_sys_modules=0*)

Import dynamically generated code as a module. code is the object containing the code (a string, a file handle or an actual compiled code object, same types as accepted by an exec statement). The name is the name to give to the module, and the final argument says wheter to add it to sys.modules or not. If it is added, a subsequent import statement using name will return this module. If it is not added to sys.modules import will try to load it in the normal fashion.

import foo

is equivalent to

foofile = open("/path/to/foo.py") foo = importCode(foofile,"foo",1)

Returns a newly generated module.

Python Module Index

r

Index

S

T