
RealTime OpenControl Documentation

Release 2.0b0

Sebastian Keller

Jul 07, 2020

1	Welcome to RealTime OpenControl's documentation	1
1.1	RTLogger backend	1
1.2	Expandable with plugins	1
1.3	Telegram-Bot	1
1.4	Websocket-Server	1
1.5	Graphical user interface (RTOC-GUI)	2
1.6	Scripting/Automation	2
2	Getting started	3
3	FAQ	5
4	Indices and tables	7
5	Table of contents	9
5.1	Installation	9
5.1.1	Installing with Python3 (recommended)	9
5.1.2	Installing from builds	9
5.1.3	Install manually	10
5.1.4	Long-time measurements in postgresSQL database (optional)	10
5.2	First steps	11
5.2.1	First run	11
5.3	Collecting data	11
5.3.1	Writing Plugins	11
5.3.2	Template.py	12
5.3.3	Plugin repository	14
5.3.3.1	Installing plugins	14
5.3.3.1.1	Manually	15
5.3.3.2	List of plugins	15
5.3.3.3	Plugin descriptions	15
5.3.3.3.1	Template	15
5.3.3.3.2	DPS5020	15
5.3.3.3.3	Generator	16
5.3.3.3.4	holdPeak_VC820	16
5.3.3.3.5	INA219_Modul	17
5.3.3.3.6	Octotouch	17
5.3.3.3.7	PIKO_Solarmodules	17

	5.3.3.3.8	System	18
	5.3.3.3.9	ReflowOfen/ReflowPlatte	18
	5.3.3.3.10	Heliotherm	19
	5.3.3.3.11	Futtertrocknung	19
5.4		Userdata	19
	5.4.1	Document-Tree	19
	5.4.2	backup (directory)	20
	5.4.3	devices (directory)	20
	5.4.4	autorun_devices	20
	5.4.5	config.json	20
	5.4.5.1	global	20
	5.4.5.2	postgresql	21
	5.4.5.3	GUI	21
	5.4.5.4	telegram	22
	5.4.5.5	websockets	22
	5.4.5.6	backup	22
	5.4.6	globalActions.json	22
	5.4.7	globalEvents.json	22
	5.4.8	plotStyles.json	23
	5.4.9	telegramActions.json	23
	5.4.10	telegram_clients.json	23
5.5		Telegram Communication	23
	5.5.1	Telegram-Bot setup	23
	5.5.1.1	Create a new Telegram-Bot	23
	5.5.1.2	Configure RTOC for Telegram	23
	5.5.1.3	User permissions	23
	5.5.1.3.1	Custom permission	24
	5.5.2	Mainmenu	25
	5.5.2.1	Latest values	25
	5.5.2.2	Signals	25
	5.5.2.3	Devices	25
	5.5.2.4	Send event/signal	25
	5.5.2.5	Automation	25
	5.5.2.6	Settings	25
	5.5.2.6.1	Set Event Notification	25
	5.5.2.6.2	General	26
	5.5.2.6.3	TelegramBot	26
	5.5.2.6.4	Backup-Settings	26
	5.5.3	Telegram Custom-menu	26
	5.5.4	User Shortcuts	27
5.6		Controlling and automation	27
	5.6.1	Overview	27
	5.6.2	Available functions and libraries	27
	5.6.2.1	Python libraries	27
	5.6.2.2	Functions to interact with RTOC	27
	5.6.2.3	Access to plugin parameters and signals in scripts	27
	5.6.2.4	Access to telegram functions	28
	5.6.2.5	Special stuff	28
	5.6.2.6	RTOC library	28
	5.6.3	Event/Action system	28
	5.6.3.1	Global actions	28
	5.6.3.2	Global events	29
5.7		Graphical user interface	30
	5.7.1	Titlebar	31

5.7.2	Device Widget	32
5.7.3	Signal Widget	32
5.7.4	Event Widget	33
5.7.5	DevicesRAW Widget	34
5.7.6	Script Widget	35
5.7.7	Run scripts in GUI	35
5.7.7.1	Trigger-System	36
5.7.8	Settings Widget	37
5.7.9	Import/Export signals/sessions	37
5.7.9.1	Import session	37
5.7.9.2	Import XLSX, MATLAB, CSV	38
5.7.10	Remote-control via Websocket	39
5.7.11	Plugin-Downloader	40
5.8	Websocket Communication	40
5.8.1	Python example (With /RTLogger/RTWebsocketClient)	42
5.9	Backend Source-code	42
5.9.1	LoggerPlugin.py	42
5.9.2	jsonsocket.py	43
5.9.3	ROTC.RTLogger Submodules	43
5.9.3.1	ROTC.RTLogger.Daemon module	43
5.9.3.2	ROTC.RTLogger.DeviceFunctions module	43
5.9.3.3	ROTC.RTLogger.EventActionFunctions module	43
5.9.3.4	ROTC.RTLogger.NetworkFunctions module	45
5.9.3.5	ROTC.RTLogger.RTLogger module	45
5.9.3.6	ROTC.RTLogger.ROTC_Web module	45
5.9.3.7	ROTC.RTLogger.ROTC_Web_standalone module	45
5.9.3.8	ROTC.RTLogger.RTRemote module	45
5.9.3.9	ROTC.RTLogger.RT_data module	45
5.9.3.10	ROTC.RTLogger.ScriptFunctions module	45
5.9.3.11	ROTC.RTLogger.importCode module	45
5.9.3.12	ROTC.RTLogger.loggerlib module	48
5.9.3.13	ROTC.RTLogger.scriptLibrary module	48
5.9.3.14	ROTC.RTLogger.telegramBot module	48
5.10	GUI - Source-code	48
5.10.1	ROTC.ROTC module	48
5.10.2	ROTC.ROTC_GUI subpackage	48
5.10.2.1	ROTC.ROTC_GUI.Actions module	48
5.10.2.2	ROTC.ROTC_GUI.RTPlotActions module	48
5.10.2.3	ROTC.ROTC_GUI.RTPlotWidget module	48
5.10.2.4	ROTC.ROTC_GUI.csvSignalWidget module	48
5.10.2.5	ROTC.ROTC_GUI.define module	48
5.10.2.6	ROTC.ROTC_GUI.eventWidget module	48
5.10.2.7	ROTC.ROTC_GUI.globalActionWidget module	48
5.10.2.8	ROTC.ROTC_GUI.globalEventWidget module	48
5.10.2.9	ROTC.ROTC_GUI.remoteWidget module	48
5.10.2.10	ROTC.ROTC_GUI.scriptHelpWidget module	48
5.10.2.11	ROTC.ROTC_GUI.scriptSubWidget module	48
5.10.2.12	ROTC.ROTC_GUI.scriptWidget module	48
5.10.2.13	ROTC.ROTC_GUI.settingsWidget module	48
5.10.2.14	ROTC.ROTC_GUI.signalEditWidget module	48
5.10.2.15	ROTC.ROTC_GUI.signalWidget module	48
5.10.2.16	ROTC.ROTC_GUI.styleMultiPlotGUI module	48
5.10.2.17	ROTC.ROTC_GUI.stylePlotGUI module	48
5.10.2.18	ROTC.ROTC.PluginDownloader module	48

5.10.2.19	RTOC.RTOC.RTOC_Import module	48
5.10.3	RTOC.lib package	48
5.10.3.1	RTOC.lib.general_lib module	48
5.10.3.2	RTOC.lib.pyqt_customlib module	48
5.11	Example: Embedded	48
5.11.1	Introduction	49
5.11.2	Prerequisite	49
5.11.3	Setup	49
5.11.4	Remote access	50
5.11.4.1	Remote-Websocket-Client (Person B)	50
5.11.4.2	Admin-Telegram-Client (Person A)	50
5.11.4.3	Custom-Telegram-Client (Person C)	50
5.11.4.4	Read-Telegram-Client (Person D)	50
5.12	Submit your plugin	50
Python Module Index		51
Index		53

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 Websocket-Server

Communication with RTLogger from other processes or devices. Suitable for embedded devices with graphical user interface. More information here: [/Websocket](#)

1.5 Graphical user interface (RTOC-GUI)

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

1.6 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 automatisations. More information here: *Controlling and automation*

CHAPTER 2

Getting started

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

See this *Example: Embedded* to get an idea of the capabilities of RTOC.

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*

- 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 *Websocket*
- 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 Websocket in the network? *Remote-control via Websocket*
- 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`

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, Websockets and Telegram. The default RTOC setup is suitable for running RTOC on embedded devices.

There are also different variations available to install:

```
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 pycopg2
```

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

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

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

```
pip3 install python-telegram-bot matplotlib
```

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
// 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

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

// 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)

    # 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."""

        self._firststrun = 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 '

```

(continues on next page)

(continued from previous page)

```

    ``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``.

        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

```

(continues on next page)

(continued from previous page)

```

    """
    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_()

    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 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*)

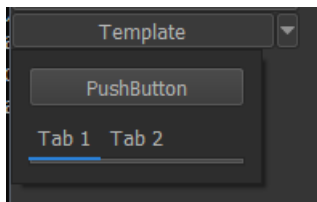
5.3.3.2 List of plugins

- 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:

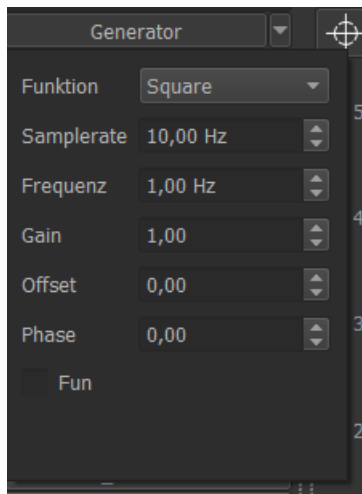
- You can set a parameters in file DPS5020.py

```
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 Generator

GUI: Yes



Dependencies: -

Target system: Each OS

Info:

5.3.3.4 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*

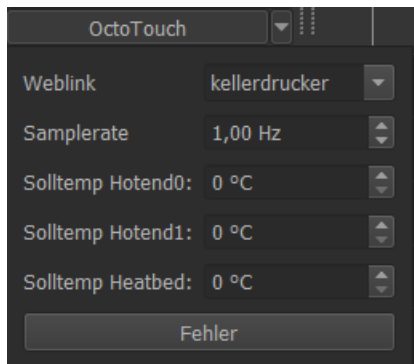
Target system: RaspberryPi (connected to INA219 via I2C)

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*

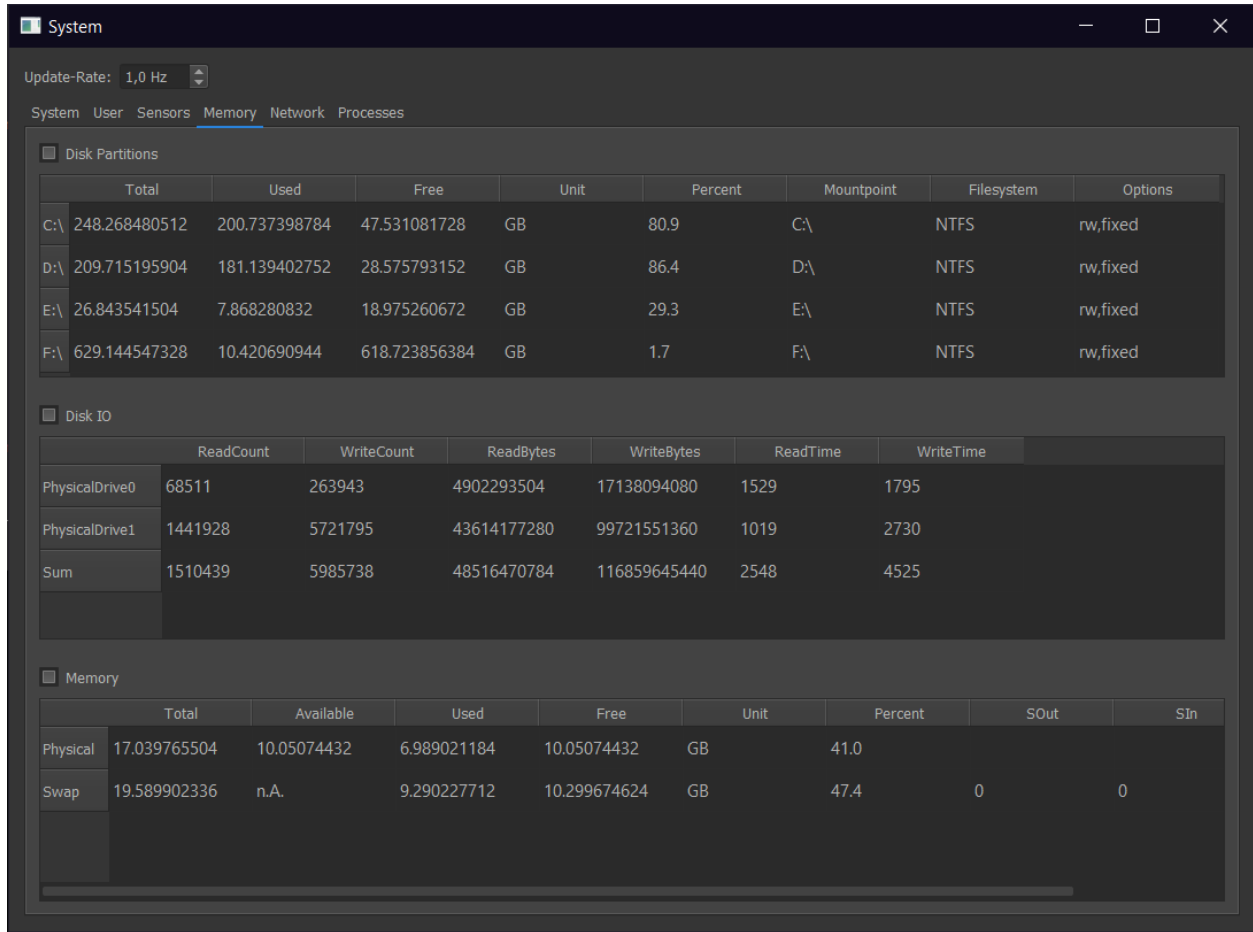
Target system: Each OS (In same network as PIKO modules)

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 addresses

5.3.3.3.8 System

GUI: Yes



Dependencies: -

Target system: Each OS

Info:

5.3.3.3.9 ReflowOfen/ReflowPlatte

GUI: Yes



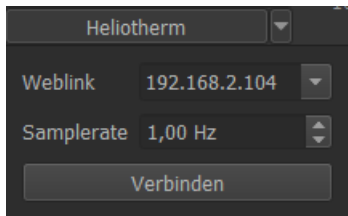
Dependencies: -

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 !
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
signalInactivityTime-out	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	De-fault	Type	Information
active	False	bool	De/activate telegram-bot
token	""	str	Your telegram bot-token
de-fault_eventlevel	0	0,1 or 2	Default eventlevel for new users
de-fault_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 websockets

Entry	Default	Type	Information
active	False	bool	De/activate Websocket-server
port	5050	int	Websocket-port
password	""	str	Optional password for Websocket-encryption (AES)
knownHosts	{ }	dict	Recent Websocket-hosts for remote connection are stored here

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:AAB-BAABB66AA11_uGG22GH44AA55_3)

5.5.1.2 Configure RTOC for Telegram

You can either 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
- **custom:** Can only call user-specific shortcuts, cannot receive event-notifications
- **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



5.5.1.3.1 Custom permission

The custom permission has a special role. A user with ‘custom’ permission will not receive notifications for events and will only see shortcuts, created for this user. Thereby you can provide a very simple bot to users, which will only be able to call selected functions or modify parameters. For example a bot with only two buttons to control the heating in the livingroom: ‘Set Temperature’ and ‘Enable/Disable’

5.5.2 Mainmenu

- <CUSTOM_ACTIONS> [**read**,**write**,**admin**]
- <USER_SHORTCUTS> [**custom**,**write**,**admin**]
- 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 bot appearance: You can switch between the inlinekeyboard-button style and chat style
- OnlyAdmin - mode: If enabled, only admins have access to the bot.
- Previews: Admins can preview the different permissions to check them.
- 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.

5.5.4 User Shortcuts

Any Telegram client can define his own shortcuts to plugin-functions and parameters. These shortcuts will be displayed in the main menu.

To define a new shortcut, you can either modify the shortcuts manually in the `telegram_clients.json` file.

Clients with 'write' or 'admin' permission can create shortcuts right in the Telegram bot. You can press the 'Create shortcut' button in every plugin-parameter or function submenu.

****Important: **** Every telegram function (`telegram_send_message,...`) inside a plugin-function will send the message, file or image only to the client, who called the shortcut.

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 Access to telegram functions

You can send messages, pictures, files and graphs with the following functions. (e.g. `telegram.send_message_to_all('Hi')`):

5.6.2.5 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`.

5.6.2.6 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
  }
}
```

(continues on next page)

(continued from previous page)

```
}
}
```

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

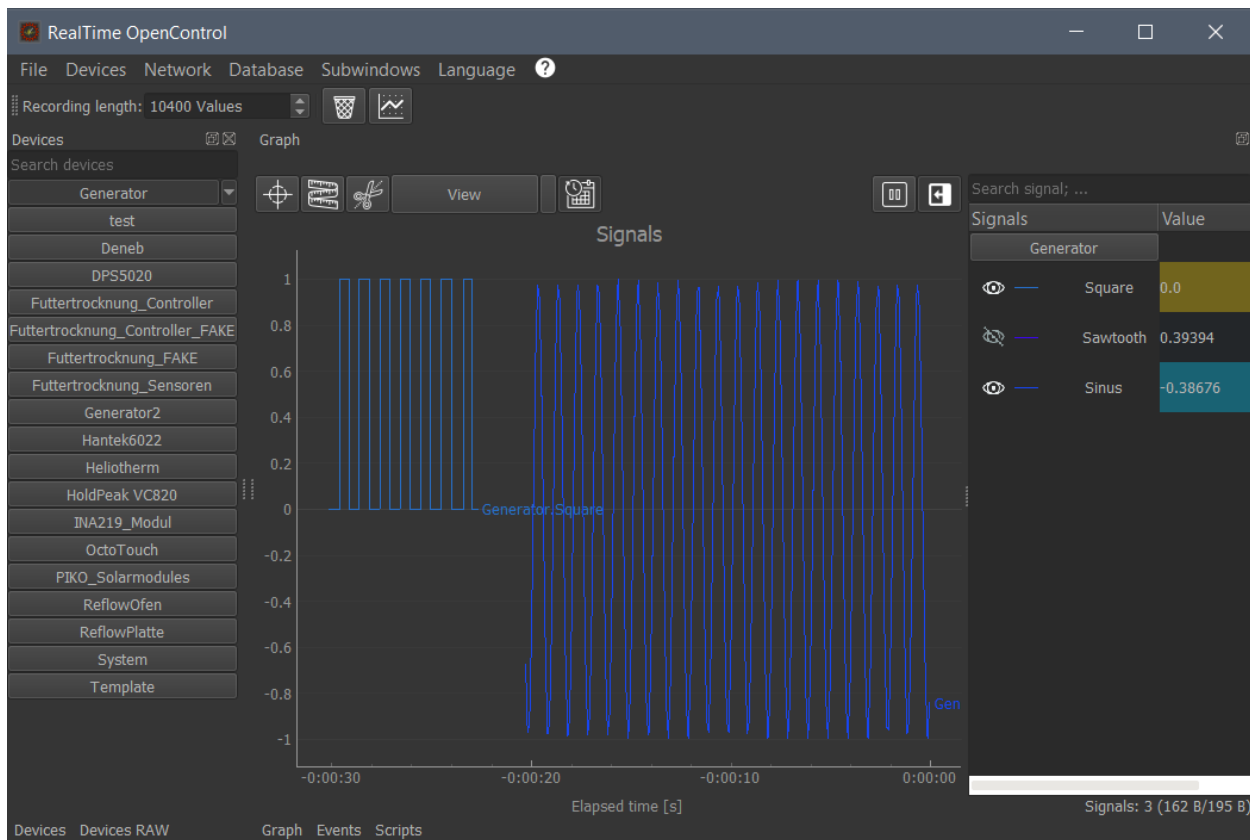
```
{
  "myevent": {
    "cond": "Generator.Square.latest >= 2",
    "text": "It is big",
    "return": " ",
    "priority": 0,
    "id": "testID",
    'trigger': 'rising',
    'sname': '',
    'dname': '',
    'active': True
  }
}
```

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 <code>LoggerPlugin.event()</code>
dname	str	...
sname	str	
priority	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 and subscribe to signals, which will then be casted to your device.

5.7.1 Titlebar

Systembar-structure



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.

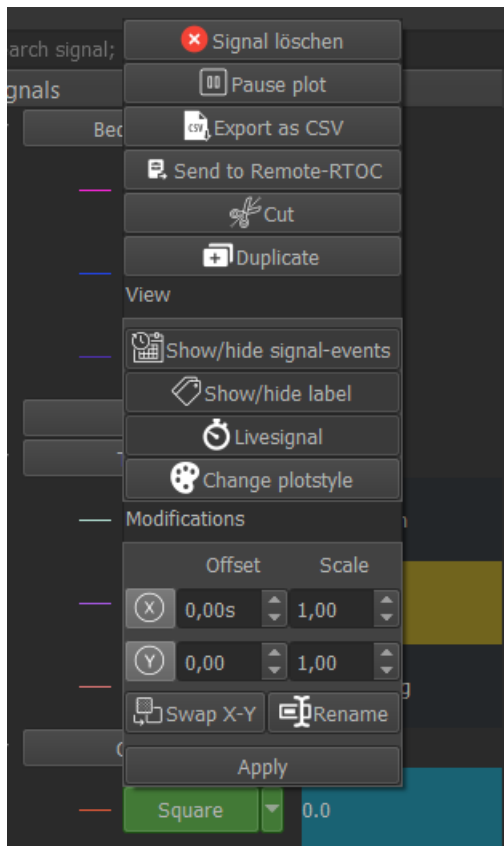
5.7.2 Device Widget

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

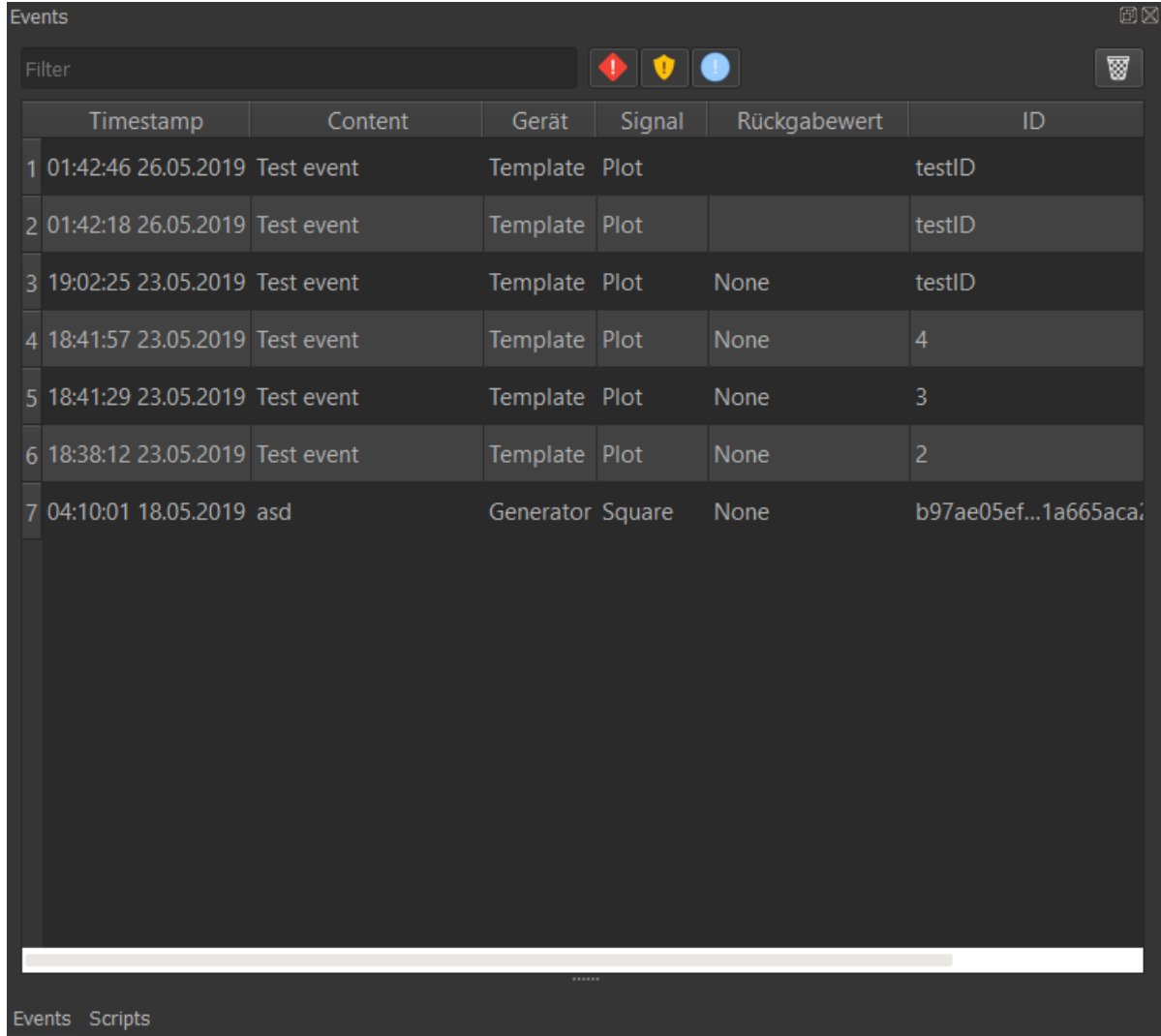
The signal widget (right) hold all signals. You can toggle each signal by clicking it (red=hidden, green=visible). Next to the button is the latest value.

Each signal also has a dropdown-menu



Check out each menu-entry by yourself, please.

5.7.4 Event Widget

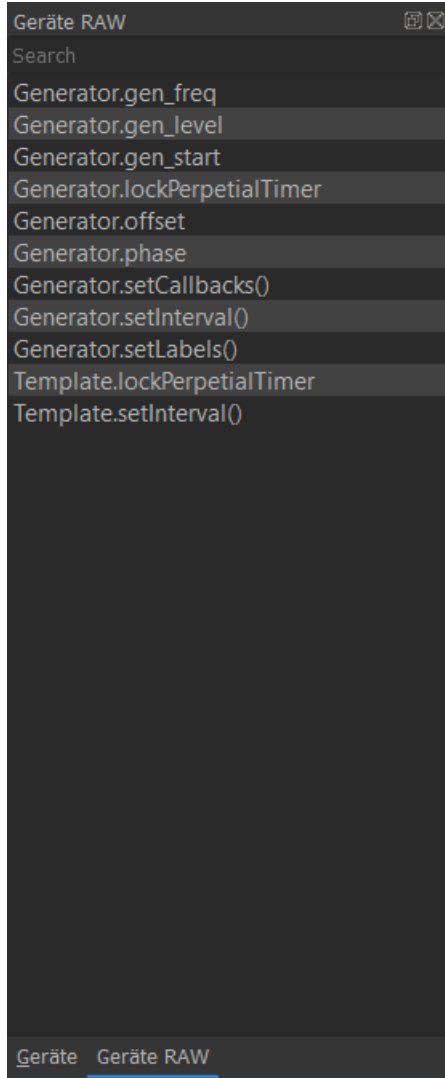


The screenshot shows the 'Events' widget interface. At the top, there is a 'Filter' input field and three priority filter buttons: a red diamond with an exclamation mark, a yellow shield with an exclamation mark, and a blue circle with an exclamation mark. To the right of these buttons is a trash can icon. Below the filter area is a table with the following columns: 'Timestamp', 'Content', 'Gerät', 'Signal', 'Rückgabewert', and 'ID'. The table contains seven rows of event data. At the bottom of the widget, there are two tabs: 'Events' (which is selected) and 'Scripts'.

	Timestamp	Content	Gerät	Signal	Rückgabewert	ID
1	01:42:46 26.05.2019	Test event	Template	Plot		testID
2	01:42:18 26.05.2019	Test event	Template	Plot		testID
3	19:02:25 23.05.2019	Test event	Template	Plot	None	testID
4	18:41:57 23.05.2019	Test event	Template	Plot	None	4
5	18:41:29 23.05.2019	Test event	Template	Plot	None	3
6	18:38:12 23.05.2019	Test event	Template	Plot	None	2
7	04:10:01 18.05.2019	asd	Generator	Square	None	b97ae05ef...1a665aca:

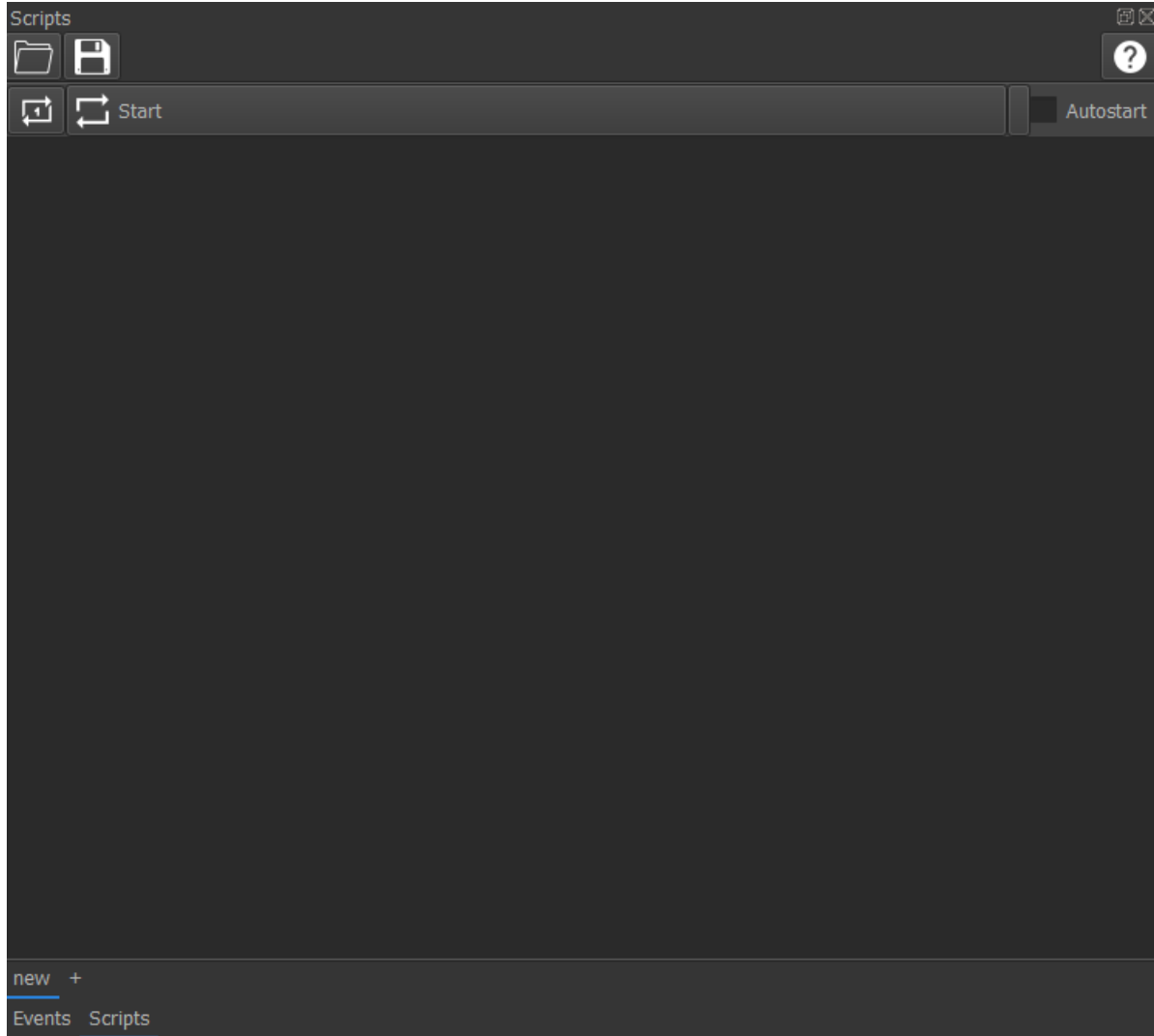
The event widget shows a list with all events. You can filter the events by text or hide specific priorities. On the right top is a button to delete all events.

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*.

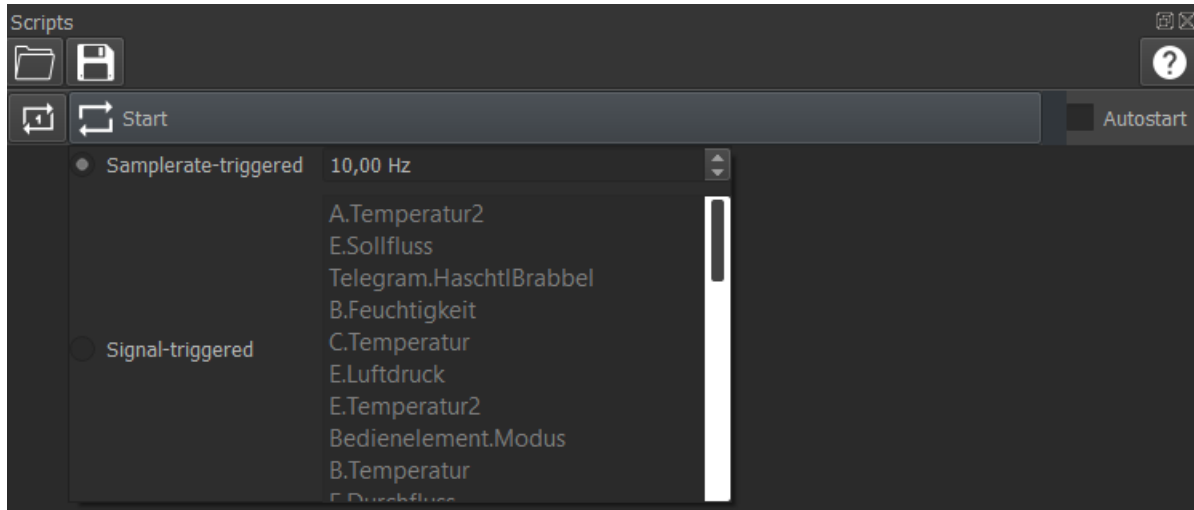
5.7.7 Run scripts in GUI

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.

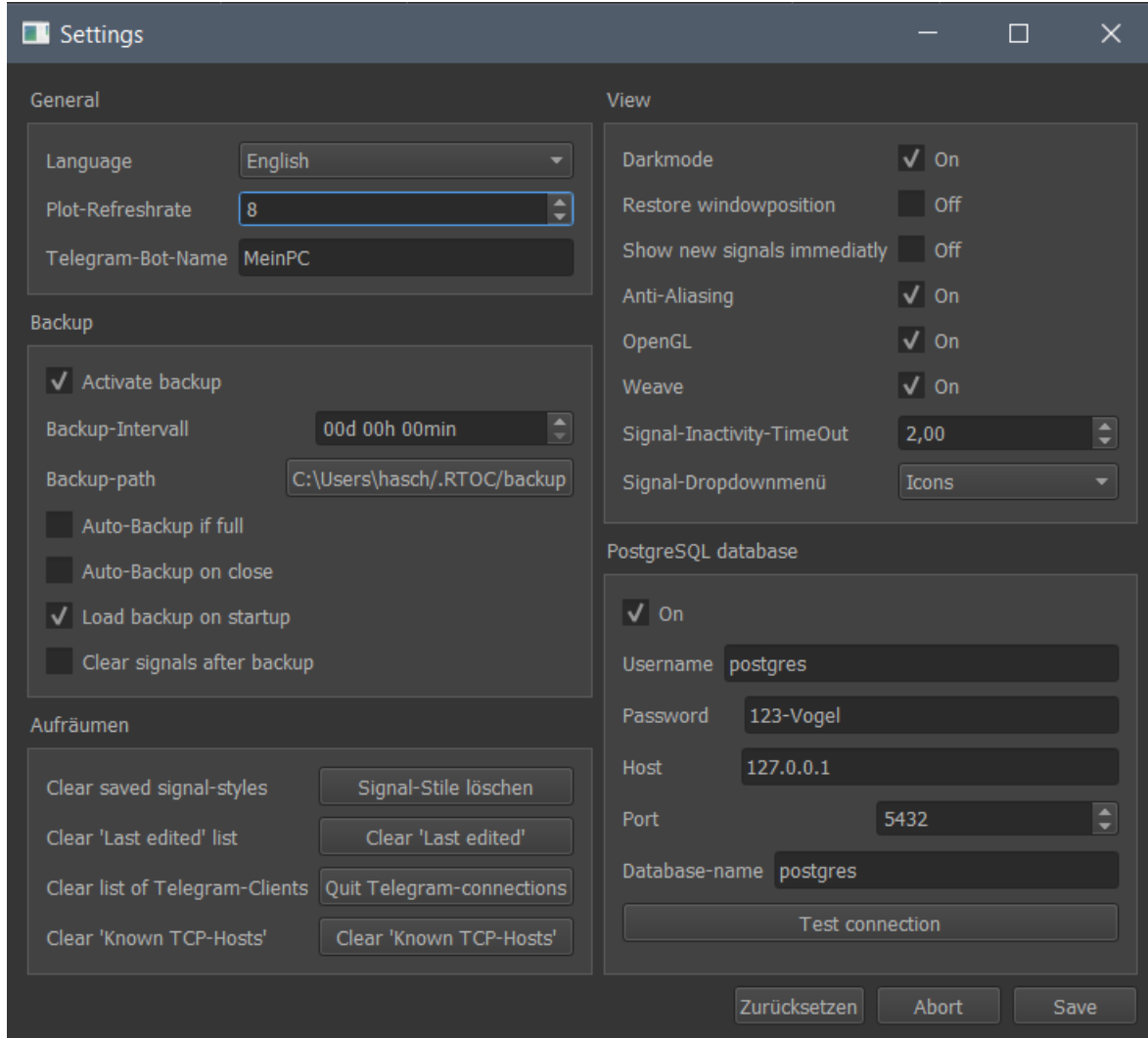
5.7.7.1 Trigger-System



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

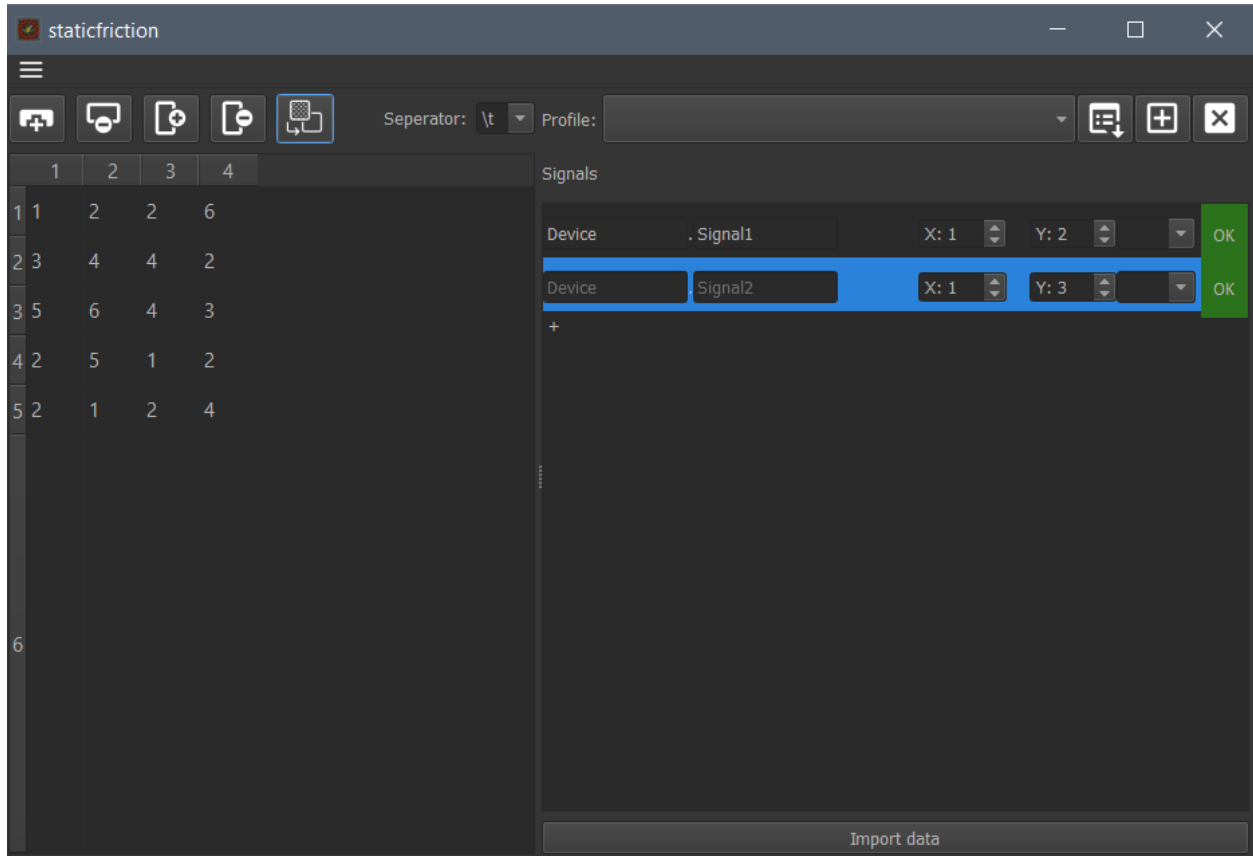
5.7.9.1 Import session

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

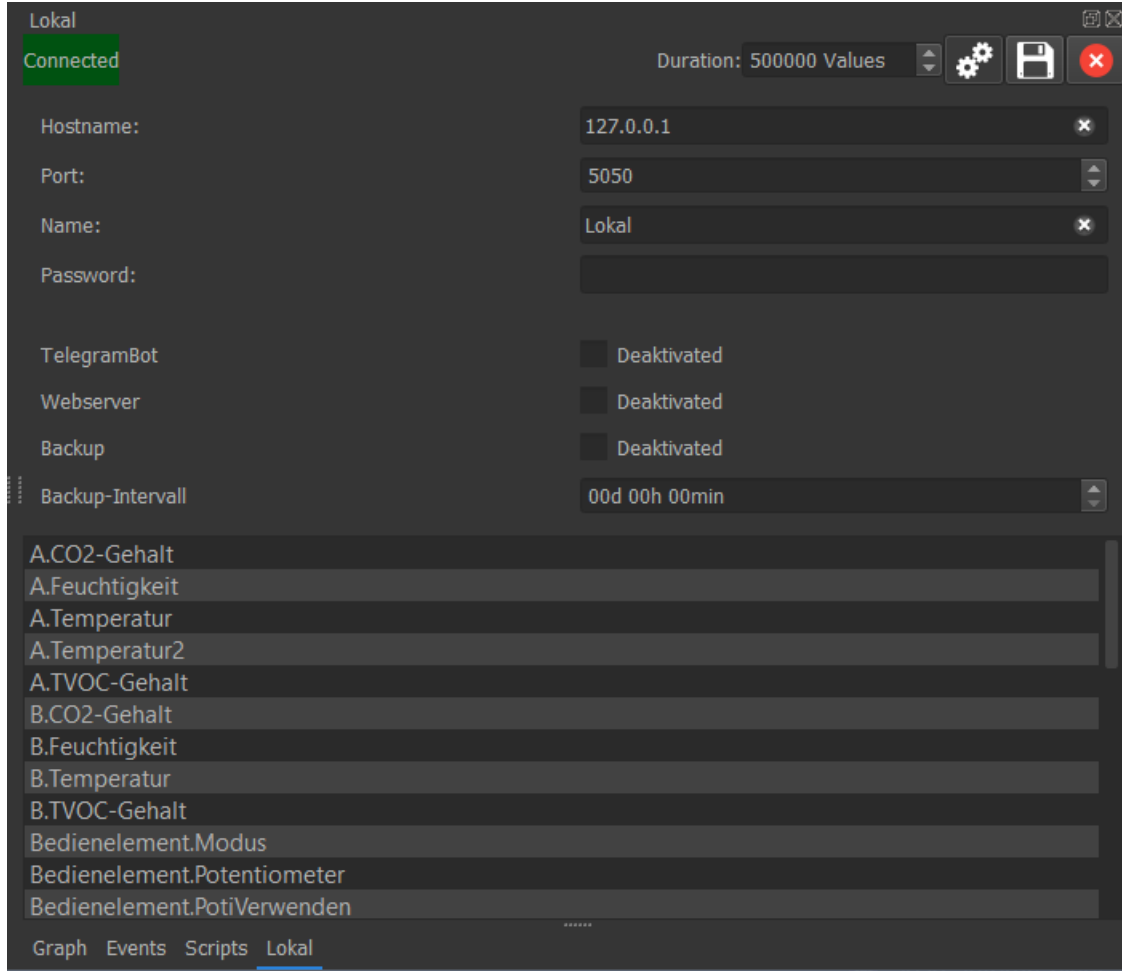
5.7.9.2 Import XLSX, MATLAB, CSV



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

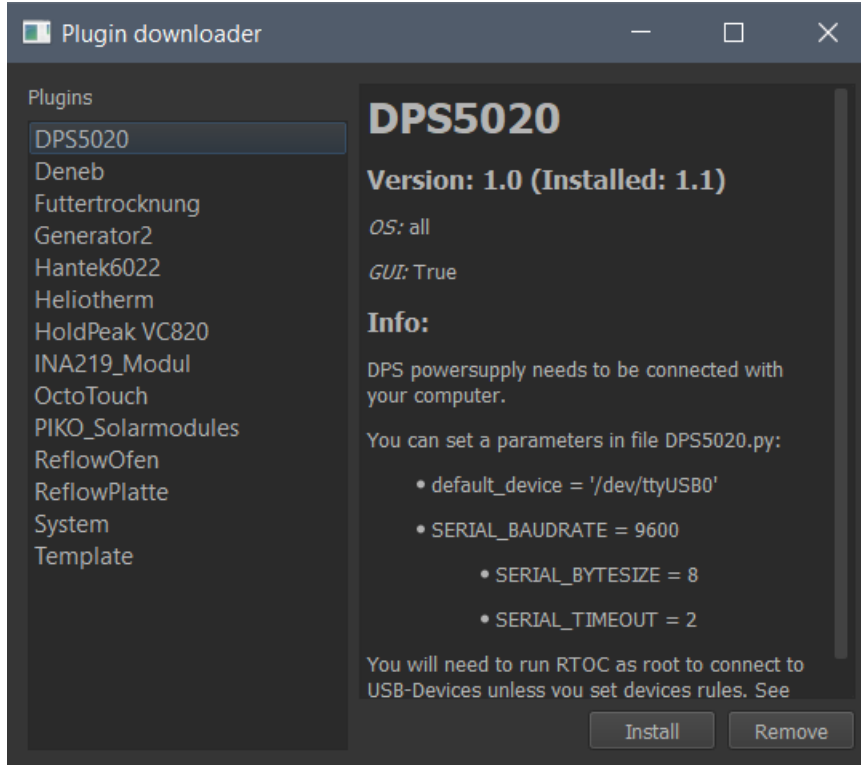
On the right side you can define signals, that will be taken from the data-table. 1. Click (+) to add a new signal. 2. Set a signal and devicename (not needed) 3. Select columns for X and Y data. If X-Column is 0, X-data will be generated automatically 4. The color on the right of each signal indicates, 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 Websocket



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 Websocket Communication

****Websocket Port: 5050, SSL: 443 ****

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

dict-key	Description
<code>“authorize “hashed_password”</code> =	You must always send this command in the beginning (If not password-protected: <code>authorize=True</code>)
<code>plot = False</code>	True: x and y data are plotted as one signal. False: x and y data are signals in pairs. See <code>LoggerPlugin.plot()</code> for more information.
<code>x = [0, 1, 2, 3, 4, 5]</code>	X data to be sent. If x is not set and <code>plot = False</code> , <code>time.time()</code> is set as x-value. If <code>plot = True</code> , the indices of the y data are used. See <code>LoggerPlugin.plot()</code> for more information.
<code>y = [1, 2, 3, 4, 5, 6]</code>	Y-data to be sent. See <code>LoggerPlugin.plot()</code> for more information.
<code>sname = ["signalname"]</code>	List of signal names with <code>plot = False</code> , only one element with <code>plot = True</code> . See <code>LoggerPlugin.plot()</code> for more information.
<code>dname = "devicename"</code>	Device name to be transmitted. See <code>LoggerPlugin.plot()</code> for more information.
<code>unit = "unit"</code>	signal unit. See <code>LoggerPlugin.plot()</code> for more information.
<code>event = {text = "", dname='', sname='', x=clock, priority=0}</code>	Create an event. See <code>LoggerPlugin.event()</code> for more information.
<code>getLatest= True</code>	If <code>getLatest=True</code> , RTOC returns a Dict of the most current measured values. The signal names are the keys. See <code>NetworkFunctions.getLatest()</code> for more information.
<code>getSignalList = True</code>	If <code>getSignalList=True</code> , RTOC returns a list of signal names. See <code>NetworkFunctions.getSignalList()</code> for more information.
<code>getEventList = True</code>	If <code>getEventList=True</code> , RTOC returns a list of all events. See <code>NetworkFunctions.getEventList()</code> for more information.
<code>getPluginList = True</code>	If <code>getPluginList =True</code> , RTOC returns a Dict of the plugins containing the plugin functions, parameters and the status of the plugin. See <code>NetworkFunctions.getPluginList()</code> for more information.
<code>getEvent = ['Device. Signal', ...]</code>	Server request for the events of a signal. See <code>NetworkFunctions.getEvent()</code> for more information.
<code>getSignal = ['Device. Signal', ...]</code>	Server request for signal data. See <code>NetworkFunctions.getSignal()</code> for more information.
<code>getSession = True</code>	
<code>remove = ['signal', 'DEVICE.SIGNAL']</code>	
<code>subscribe = ['signal', 'DEVICE. SIGNAL']</code>	
<code>unsubscribe = ['signal', 'DEVICE. SIGNAL']</code>	
<code>subscribeAll = True</code>	
<code>unsubscribeAll = True</code>	
<code>plugin = {...}</code>	Access to plugins with Dict. See <code>NetworkFunctions.handleTcpPlugins()</code> for more information.
<code>logger = {...}</code>	RTOC default functions. See <code>/RTLogger/RTWebsocketServer()</code> for more information.
<code>userAction = {...}</code>	
<code>automation = {...}</code>	

As response RTOC delivers a dict with the following keys:

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

5.8.1 Python example (With /RTLogger/RTWebsocketClient)

This example uses the module /RTLogger/RTWebsocketClient:

```
import RTWebsocketClient

data = {'x':[0,1,2,3], 'y':[1,2,3,4], 'dname':'Test', 'sname':['T1', 'T2', 'T3', 'T4']}
sock = RTWebsocketClient.RTWebsocketClient()
sock.on_message = lambda msg: print(msg)
sock.connect('127.0.0.1', 5050)
sock.send(data)
input('Press enter to quit')
sock.close()
```

Have a look into the RTWebsocketClient-File to see, which websocket-callbacks are available.

5.9 Backend Source-code

5.9.1 LoggerPlugin.py

This class is imported by any plugin written for RTOC.

It includes all functions to interact with RTOC. Every plugin must inherit this class! Your plugin must look like this

```
::

from RTOC.LoggerPlugin import LoggerPlugin

class Plugin(LoggerPlugin):
    def __init__(self, *args, **kwargs):
        LoggerPlugin.__init__(self, *args, **kwargs)
    ...
    ...
```

If you need to pass arguments to your class initialization, use kwargs. ('stream', 'plot', 'event', 'telegramBot' cannot be used as kwarg-name)

5.9.2 jsonsocket.py

5.9.3 RTOC.RTLogger Submodules

5.9.3.1 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*)

Bases: object

A generic daemon class.

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

daemonize ()

Daemonize 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.

5.9.3.2 RTOC.RTLogger.DeviceFunctions module

5.9.3.3 RTOC.RTLogger.EventActionFunctions module

class EventActionFunctions

Bases: object

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)

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

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

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 fulfilled. 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*)

resetGlobalEventState ()

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*)

5.9.3.4 RTOC.RTLogger.NetworkFunctions module

5.9.3.5 RTOC.RTLogger.RTLogger module

5.9.3.6 RTOC.RTLogger.RTOC_Web module

5.9.3.7 RTOC.RTLogger.RTOC_Web_standalone module

5.9.3.8 RTOC.RTLogger.RTRemote module

5.9.3.9 RTOC.RTLogger.RT_data module

5.9.3.10 RTOC.RTLogger.ScriptFunctions module

class ScriptFunctions

Bases: object

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*)

replaceTelegramFunctions (*s*)

triggerExpressionHandler (*expression*)

5.9.3.11 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.

5.9.3.12 RTOC.RTLogger.loggerlib module

5.9.3.13 RTOC.RTLogger.scriptLibrary module

5.9.3.14 RTOC.RTLogger.telegramBot module

5.10 GUI - Source-code

5.10.1 RTOC.RTOC module

5.10.2 RTOC.RTOC_GUI subpackage

5.10.2.1 RTOC.RTOC_GUI.Actions module

5.10.2.2 RTOC.RTOC_GUI.RTPlotActions module

5.10.2.3 RTOC.RTOC_GUI.RTPlotWidget module

5.10.2.4 RTOC.RTOC_GUI.csvSignalWidget module

5.10.2.5 RTOC.RTOC_GUI.define module

5.10.2.6 RTOC.RTOC_GUI.eventWidget module

5.10.2.7 RTOC.RTOC_GUI.globalActionWidget module

5.10.2.8 RTOC.RTOC_GUI.globalEventWidget module

5.10.2.9 RTOC.RTOC_GUI.remoteWidget module

5.10.2.10 RTOC.RTOC_GUI.scriptHelpWidget module

5.10.2.11 RTOC.RTOC_GUI.scriptSubWidget module

5.10.2.12 RTOC.RTOC_GUI.scriptWidget module

5.10.2.13 RTOC.RTOC_GUI.settingsWidget module

5.10.2.14 RTOC.RTOC_GUI.signalEditWidget module

5.10.2.15 RTOC.RTOC_GUI.signalWidget module

5.10.2.16 RTOC.RTOC_GUI.styleMultiPlotGUI module

5.10.2.17 RTOC.RTOC_GUI.stylePlotGUI module

5.10.2.18 RTOC.RTOC.PluginDownloader module

5.10.2.19 RTOC.RTOC.RTOC_Import module

5.10.3 RTOC.lib package

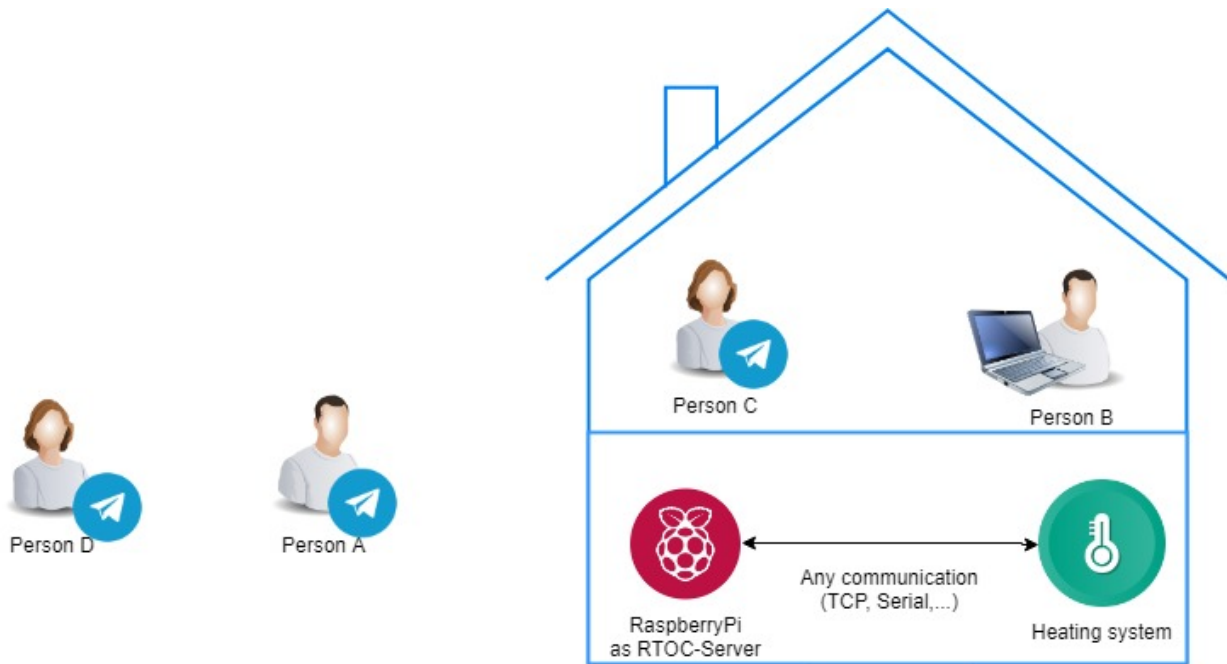
5.10.3.1 RTOC.lib.general lib module

5.10.3.2 RTOC.lib.pyqt_customlib module

5.11 Example: Embedded

- PostgreSQL-database for long-time logging
- Telegram-Bot with different permissions
- Remote-Websocket connection
- Automation (Event-Action-System)

5.11.1 Introduction



This example will show, how to connect a heatpump or any other heating system with an RTOC-server. This will allow remote control and long-time-measurements. (And events,... all RTOC-Features)

Person A and B are administrators of the RTOC-server. Person A from outside the local network using the Telegram-Bot and Person B in the same network with the RTOC-server using a PC.

Person C is a user, who just wants to control the heating system (Smart Home). He/she will be able to set the room-temperature or any other custom functions with the telegramBot.

Person D is just interested in your heating's long-time-measurements (e.g. a technician) and has access with the telegramBot.

5.11.2 Prerequisite

- Raspberry Pi
- Heating system/heat pump with remote control capability
- RTOC-Plugin to interact with the heating system (e.g. Heliotherm from RTOC_repo)

5.11.3 Setup

1. Install RTOC: *Installation*

2. Setup PostgreSQL Database: *Long-time measurements in postgresQL database (optional)*
3. Setup Telegram-Bot: *Telegram-Bot setup*
4. You should also set a websocket-password in *config.json*
5. Download/program a plugin. Refer to *Plugin repository* and *Writing Plugins*.
6. Add the plugin to *autorun_devices*.
7. Start RTOC with *python3 -m RTOC*. Use *loggerServer.py* to run RTOC as a service on linux.

5.11.4 Remote access

5.11.4.1 Remote-Websocket-Client (Person B)

Person B uses a computer with RTOC installed. Have a look at the *Graphical user interface*-Documentation for more information. He can view all of the data in the database of the RTOC-server and has access to all functions and parameters from the heating system. He can manipulate the data with the RTOC-GUI. Person B can instead also use the RTOC_Remote Android App

5.11.4.2 Admin-Telegram-Client (Person A)

Person A uses the Telegram-Bot to access the heating system. He has also access to all functions and parameters from the heating system and can view and edit all of the data. He can also modify the event-action (automation).

5.11.4.3 Custom-Telegram-Client (Person C)

Person C just wants to set the room temperature for example. But does not want to use the complex structure of the Telegram-Bot to do this (Devices->Heatpump->Functions->SetTemp). Therefore the telegram administrators can set the permissions for Person C to 'custom'. Then Person C will only see shortcuts, which have been created for only this user. This can be done in two ways: 1. Set the permission of Person C to 'Write' and create shortcuts for the desired functions/parameters and change the permission to 'Custom' afterwards. 2. Edit the client-shortcuts in *telegram_clients.json*.

5.11.4.4 Read-Telegram-Client (Person D)

Person D just wants to view the measurements. Therefore an administrator can set his permission to 'Read'.

5.12 Submit your plugin

If you want to publish your plugin in my repository, send me an email: kellersebastian@protonmail.com

r

RTOC.RTLogger.Daemon, 43
RTOC.RTLogger.EventActionFunctions, 43
RTOC.RTLogger.importCode, 45
RTOC.RTLogger.ScriptFunctions, 45
RTOC.RTOC_GUI.define, 48

A

addGlobalAction() (*EventActionFunctions method*), 43
 addGlobalEvent() (*EventActionFunctions method*), 43

C

checkCondition() (*ScriptFunctions method*), 45
 checkGlobalActions() (*EventActionFunctions method*), 43
 checkGlobalEvents() (*EventActionFunctions method*), 43
 createConditionFunction() (*ScriptFunctions method*), 45
 createFunction() (*ScriptFunctions method*), 45

D

Daemon (*class in RTOC.RTLogger.Daemon*), 43
 daemonize() (*Daemon method*), 43
 delpid() (*Daemon method*), 43

E

editGlobalAction() (*EventActionFunctions method*), 44
 editGlobalEvent() (*EventActionFunctions method*), 44
 EventActionFunctions (*class in RTOC.RTLogger.EventActionFunctions*), 43
 executeScript() (*ScriptFunctions method*), 45

G

generateCode() (*ScriptFunctions method*), 45
 generateTriggerCode() (*ScriptFunctions method*), 45

I

importCode() (*in RTOC.RTLogger.importCode*), 45

L

loadGlobalActions() (*EventActionFunctions method*), 44
 loadGlobalEvents() (*EventActionFunctions method*), 44

P

performGlobalActions() (*EventActionFunctions method*), 44
 performGlobalEvents() (*EventActionFunctions method*), 44
 printfunction() (*ScriptFunctions method*), 45
 printGlobalActions() (*EventActionFunctions method*), 44
 printGlobalEvents() (*EventActionFunctions method*), 44

R

removeGlobalAction() (*EventActionFunctions method*), 44
 removeGlobalEvent() (*EventActionFunctions method*), 44
 replaceGlobalVariables() (*ScriptFunctions method*), 45
 replaceLibraryFunctions() (*ScriptFunctions method*), 45
 replaceLoggerFunctions() (*ScriptFunctions method*), 45
 replacePluginMethods() (*ScriptFunctions method*), 45
 replacePluginParameters() (*ScriptFunctions method*), 45
 replaceSignalNames() (*ScriptFunctions method*), 45
 replaceTelegramFunctions() (*ScriptFunctions method*), 45
 resetGlobalEventState() (*EventActionFunctions method*), 44
 restart() (*Daemon method*), 43

RTOC.RTLogger.Daemon (*module*), 43
RTOC.RTLogger.EventActionFunctions (*module*), 43
RTOC.RTLogger.importCode (*module*), 45
RTOC.RTLogger.ScriptFunctions (*module*), 45
RTOC.RTOC_GUI.define (*module*), 48
run () (*Daemon method*), 43

S

saveGlobalActions () (*EventActionFunctions method*), 44
saveGlobalEvents () (*EventActionFunctions method*), 44
ScriptFunctions (*class in RTOC.RTLogger.ScriptFunctions*), 45
start () (*Daemon method*), 43
stop () (*Daemon method*), 43

T

translate () (*in module RTOC.RTLogger.EventActionFunctions*), 44
triggerExpressionHandler () (*ScriptFunctions method*), 45
triggerGlobalAction () (*EventActionFunctions method*), 44
triggerGlobalEvent () (*EventActionFunctions method*), 44