

ICOc Documentation

MyTooliT

Contents

1	ICOc	2
1.1	Requirements	3
1.2	Install	6
1.3	Basic Usage	8
1.4	Measurement Data	9
2	Tutorials	15
2.1	Sensor Device Renaming	15
2.2	Command Line Usage of IC Oc	16
2.3	Production Tests	20
2.4	Verification Tests	22
2.5	Virtualization	24
3	Scripts	30
3.1	EEPROM Check	30
3.2	IC Oc	30
3.3	MAC Address Conversion	30
3.4	Remove Log, HDF and PDF Files	31
3.5	Test-STH	31
3.6	Test-STU	31
3.7	Test-SMH	31
4	Development	32
4.1	Style	32
4.2	Tests	32
4.3	Release	36

5	Releases	37
5.1	Version 1.6.0	37
5.2	Version 1.5.0	37
5.3	Version 1.4.0	38
5.4	Version 1.3.0	40
5.5	Version 1.2.0	40
5.6	Version 1.1.0	41
5.7	Version 1.0.14	43
5.8	Version 1.0.13	43
5.9	Version 1.0.12	45
5.10	Version 1.0.11	46
5.11	Version 1.0.10	48
5.12	Version 1.0.9	50
5.13	Version 1.0.8	51
5.14	Version 1.0.7	51
5.15	Version 1.0.6	52
5.16	Version 1.0.5	52
5.17	Version 1.0.4	53
5.18	Version 1.0.3	54
5.19	Version 1.0.2	54
5.20	Version 1.0.0	55

1 ICOC

ICOC is a collection of tools and scripts for the ICOTronic system. Currently the main purpose of the software is

- **data collection** (via the script `icoc`) and
- **testing** the functionality of
 - Stationary Transceiver Unit (STU) and
 - sensor devices/nodes, such as
 - * Sensory Holder Assembly (SHA)/Sensory Tool Holder (STH)
 - * Sensory Milling Head (SMH)

The software reads data from the Stationary Transceiver Unit (STU) via CAN using the MyTooliT protocol. The STU itself reads from and writes data to the sensor devices via Bluetooth.

The framework currently requires

- Microsoft Windows 10, and
- Python 3.

Notes:

- In theory you can also use ICOC in Windows 11. However, we did not test the software on this operating system.
- The test suite (which uses a CAN class based on python-can) also works on Linux and macOS. The ICOC measurement software does **not**. For more information on how to use (parts of) the ICOC software on Linux, please take a look here.

For more information about other required software components, please read the subsection “Software” in this document.

1.1 Requirements

1.1.1 Hardware

In order to use ICOC you need at least:

- a PCAN adapter:



Figure 1: PCAN Adapter

including:

- power injector, and



Figure 2: Power Injector

– power supply unit (for the power injector):

- a Stationary Transceiver Unit:
- a sensor device, such as a Sensory Tool Holder:

1.1.1.1 Setup

1. Connect the power injector
 1. to the PCAN adapter, and
 2. the power supply unit.
2. Connect the USB connector of the PCAN adapter to your computer.
3. Make sure that your sensor device (SHA/STH/SMH) is connected to a power source. For an STH this usually means that you should check that the battery is (fully) charged.

1.1.2 Software

1.1.2.1 Python ICOC requires at least Python 3.7. The software also supports Python 3.8, 3.9 and 3.10. You can download Python [here](#). When you install the software, please do not forget to enable the checkbox “**Add Python to PATH**” in the setup window of the installer.

1.1.2.2 PCAN Driver To communicate with the STU you need a driver that works with the PCAN adapter. The text below describes how to install/enable this driver on

- Linux,
- macOS, and
- Windows.

1.1.2.2.1 Linux You need to make sure that your CAN adapter is available via the SocketCAN interface. The following steps describe one possible option to configure the CAN interface on Fedora Linux **manually**.

1. Connect the CAN adapter to the computer that runs Linux (or alternatively the Linux VM)
2. Check the list of available interfaces:

```
networkctl list
```

The command output should list the CAN interface with the name `can0`

3. Configure the CAN interface with the following command:

```
sudo ip link set can0 type can bitrate 1000000
```

4. Bring up the CAN interface

```
sudo ip link set can0 up
```

You can also bring up the CAN interface **automatically**. For that please store the following text:

```
[Match]
Name=can*

[CAN]
BitRate=1000000
```

in a file called `/etc/systemd/network/can.network`. After that you can either restart your computer/VM or reload the configuration with the commands:

```
sudo systemctl restart systemd-networkd
networkctl reload
```

Sources:

- SocketCAN device on Ubuntu Core
- Question: How can I automatically bring up CAN interface using netplan?
- networkd › systemd › Wiki › ubuntuusers

1.1.2.2.2 macOS On macOS you can use the PCUSB library to add support for the PCAN adapter. For more information on how to install this library please take a look [here](#).

1.1.2.2.3 Windows You can find the download link for the PCAN Windows driver [here](#). Please make sure that you include the “PCAN-Basic API” when you install the software.

1.1.2.3 Simplicity Commander (Optional) For the tests that require a firmware flash you need to **either** install

- Simplicity Studio or
- Simplicity Commander.

If you choose the first option, then please make sure to install the Simplicity Commander tool inside Simplicity Studio.

If you download Simplicity Commander directly, then the tests assume that you unzipped the files into the directory `C:\SiliconLabs\Simplicity Commander` on Windows.

- If you do not use the standard install path on **Windows**, then please add the path to `commander.exe` to the list `COMMANDS → PATH → WINDOWS` in the configuration file `config.yaml`.
- If you use **Linux**, then please add the path to `commander` to the list `COMMANDS → PATH → LINUX`.
- If you install Simplicity Studio in the standard install path on **macOS** (`/Applications`) you do not need to change the config. If you
 - put the application in a different directory or
 - installed Simplicity Commander directly

then please add the path to `commander` to the list `COMMANDS → PATH → MAC`.

If you do not want to change the config file, then please just make sure that `commander` is accessible via the `PATH` environment variable.

Please note, that you do not need to install Simplicity Commander if you just want to measure data with IC0c.

1.2 Install

1.2.1 Clone the Repository

Please clone this repository to a directory of your choice. You can either use the command line tool `git`:

```
git clone https://github.com/MyToolIT/IC0c.git
```

or one of the many available graphical user interfaces for Git to do that.

1.2.1.1 Example: Clone the Repository With GitHub Desktop

1. Download and install Github Desktop
2. Open Github Desktop and log into your GitHub account (optional)
3. Open the repository website in your browser
4. Click on the green button “Code” in your browser window
5. Select the item “Open with GitHub Desktop”
6. Inside GitHub Desktop choose the path where you want to clone the repository (button “Choose...”) e.g. `Documents/IC0c` in your home directory
7. Press the button “Clone”
8. Wait until the cloning process is finished

1.2.2 Install the Python Package

Before you use the software you need to install it (in developer mode). To do that please open a terminal window in the directory where you cloned the repository and run the following command:

```
pip install -e .
```

Afterwards you can use the various scripts included in the package.

1.2.2.1 Example: Install the Package Using Windows Terminal

1. Install (Windows) Terminal if you have not done so already; On Windows 11 this application should be installed by default.
2. Open Terminal
3. Enter the text `cd` (**including a trailing space character**) in the Terminal window
4. Go to the directory where you cloned the IC0c repository (e.g. `Documents/IC0c`) in File Explorer and “copy” (Ctrl + C) the directory
5. “Paste” the copied directory into the Terminal window (Ctrl + V)
6. The resulting text in the Terminal now should look something like this (if you cloned the directory to `Documents/IC0c` in your home directory)

```
cd C:\Users\USERNAME\Documents\IC0c
```

7. Press Return
8. Copy and paste the following text into the Terminal

```
pip install -e .
```

9. Press Return
10. Wait until the installation finished successfully
11. Close Terminal

1.2.3 Troubleshooting

1.2.3.1 Import Errors If one of the tests or IC0c fails with an error message that looks similar to the following text:

Traceback (most recent call last):

...

```
from numexpr.interpreter import MAX_THREADS, use_vml, __BLOCK_SIZE1__  
ImportError: DLL load failed while importing interpreter: The specified module could not be found.
```

DLL load failed while importing interpreter: The specified module could not be found.

then you probably need to install the “Microsoft Visual C++ Redistributable package”. You can download the latest version (for the x64 architecture) [here](#).

1.2.3.2 Insufficient Rights If you do not have sufficient rights to install the package you can also try to install the package in the user folder:

```
pip install --user -e .
```

The command above might not work on Linux due to a bug in `pip`. In that case you can try the following workaround to install IC0c:

```
python3 -m pip install --prefix=$(python3 -m site --user-base) -e .
```

1.2.3.3 Unknown Command icoc If `pip install` prints warnings about the path that look like this:

The script ... is installed in '...\Scripts' which is not on PATH.

then please add the text between the single quotes (without the quotes) to your PATH environment variable. Here `...\Scripts` is just a placeholder. Please use the value that `pip install` prints on your machine. If you used the installer from the Python website (and checked “Add Python to PATH”) or you used winget to install Python, then the warning above should not appear. On the other hand, the **Python version from the Microsoft Store might not add the Scripts directory** to your path.

1.3 Basic Usage

1.3.1 Starting the Program

The IC0c script can be used to control a sensor device. After you enter the command

```
icoc
```

in your terminal, a text based interface shows you the currently available options. For example, the text

```
IC0c
-----
      Name      Address      RSSI
-----
1: Blubb      08:6b:d7:01:de:81  -44 dBm

1-9: Connect to STH

f: Change Output File Name
n: Change STH Name

q: Quit IC0c
```

shows that currently one sensor device was detected. The

- Bluetooth MAC address of the device is `08:6b:d7:01:de:81`, while its
- advertisement name is “Blubb”.

The last value “-44” is the current received signal strength indication (RSSI). To exit the program use the key `q`.

1.3.2 Reading Acceleration Data

To read data from an STH (or SHA), start the ICOC script, and connect to an STH. To do that, enter the number in front of an STH entry (e.g. 1 for the first detected STH) and use the return key to confirm your selection. The text based interface will now show you something like this:

```
ICOC
STH "Blubb" (08:6b:d7:01:de:81)
-----

Hardware Version      1.4.0
Firmware Version     2.1.10
Firmware Release Name Tanja
Serial Number        -

Supply Voltage        3.16 V
Chip Temperature      26.2 °C

Run Time              0 s

Prescaler             2
Acquisition Time      8
Oversampling Rate     64
  Sampling Rate       9524
Reference Voltage      VDD

Sensors               X

s: Start Data Acquisition

n: Change STH Name
r: Change Run Time
a: Configure ADC
p: Configure Sensors
0: Set Standby Mode

q: Disconnect from STH
```

To start the data acquisition press the key s. Afterwards a graphical window

will show the measured acceleration. To stop the data acquisition, click the close button on the top of the graph. For more information on how to use ICOC and the test suite, please take a look at the section “Tutorials”.

1.4 Measurement Data

The ICOC script stores measured acceleration values in HDF5 files. By default these files will be stored in the root of the repository with a

- name starting with the text **Measurement**
- followed by a date/time-stamp,

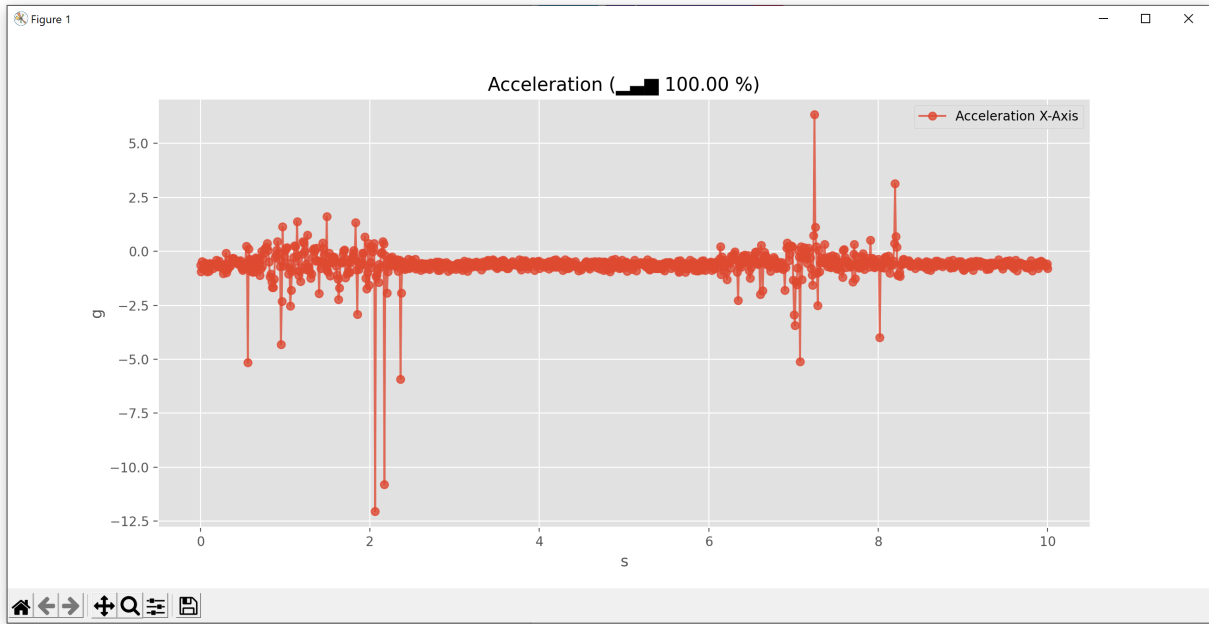


Figure 3: Acceleration

- and the extension `.hdf5`.

To take a look at the measurement data you can use the tool HDFView. Unfortunately you need to create a free account to download the program. If you do not want to register, then you can try if one of the accounts listed at BugMeNot works.

The screenshot below shows a measurement file produced by ICOc:

As you can see the table with the name `acceleration` stores the acceleration data. The screenshot above displays the metadata of the table. The most important meta attributes here are probably:

- `Start_Time`, which contains the start time of the measurement run in ISO format, and
- `Sensor_Range`, which specifies the range of the used acceleration sensor in multiples of earth's gravitation ($g \approx 9.81 \text{ m/s}^2$).

After you double click on the acceleration table on the left, HDFView will show you the actual acceleration data:

As you can infer from the `x` column above the table shows the acceleration measurement data (in multiples of g) for a single axis. The table below describes the meaning of the columns:

Column	Description	Unit
counter	A cyclic counter value (0–255) sent with the acceleration data to recognize lost packets	–
timestamp	The timestamp for the measured value in microseconds since the measurement start	s
x	Acceleration in the x direction as multiples of earth's gravitation	g ($\approx 9.81 \text{ m/s}^2$)

Depending on your sensor and your settings the table might also contain columns for the y and/or z axis.



Figure 4: Main Window of HDFView



Figure 5: Acceleration Table in HDFView

If you want you can also use HDFView to print a simple graph for your acceleration data. To do that:

1. Select the values for the the ordinate (e.g. click on the x column to select all acceleration data for the x axis)
2. Click on the graph icon in the top left corner
3. Choose the data for the abscissa (e.g. the timestamp column)
4. Click on the “OK” button

The screenshot below shows an example of such a graph:

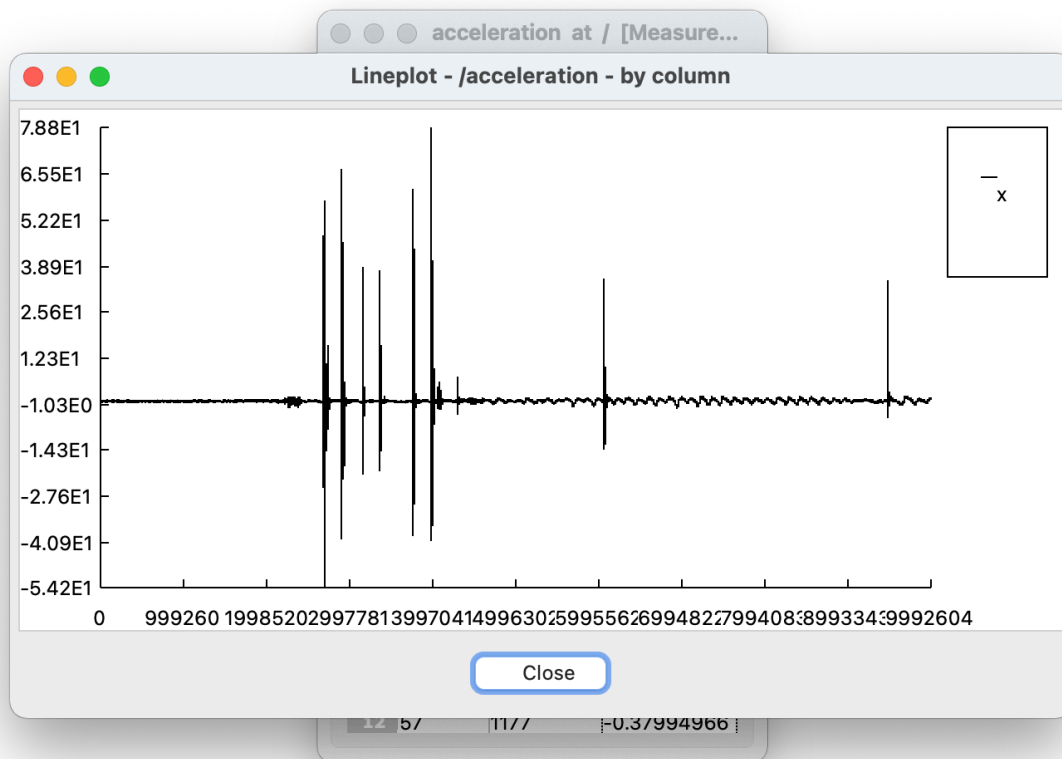


Figure 6: Acceleration Graph in HDFView

1.4.1 Adding Custom Metadata

Sometimes you also want to add additional data about a measurement. To do that you can also use HDFView. Since the tool opens files in read-only mode by default you need to change the default file access mode to “Read/Write” first:

1. Open HDFView
2. Click on “Tools” → “User Options”
3. Select “General Settings”
4. Under the text “Default File Access Mode” choose “Read/Write”
5. Close HDFView

Now you should be able to add and modify attributes. For example, to add a revolutions per minute (RPM) value of 15000 you can use the following steps:

1. Open the measurement file in HDFView
2. Click on the table “acceleration” in the left part of the window
3. In the tab “Object Attribute Info” on the right, click on the button “Add attribute”
4. Check that “Object List” contains the value “/acceleration”
5. Enter the text “RPM” in the field “Name”
6. In the field “Value” enter the text “15000”
7. The “Datatype Class” should be set to “INTEGER”
8. For the size (in bits) choose a bit length that is large enough to store the value. In our example everything equal to or larger than 16 bits should work.
9. Optionally you can also check “Unsigned”, if you are sure that you only want to store positive values
10. Click the button “OK”



Figure 7: HDFView: RPM Attribute

Sometimes you also want to add some general purpose data. For that you can use the “STRING” datatype class. For example, to store the text “hello world” in an attribute called “Comment” you can do the following

1. Repeat steps 1. – 4. from above
2. Choose “STRING” as “Datatype Class”
3. Under “Array Size” choose a length that is large enough to store the text such as “1000” (every size larger than or equal to 11 characters should work)
4. Click the button “OK”

If you want you can also add multiline text. Since you can not add newlines using in HDFView directly, we recommend you open your favorite text editor to write the text and then copy and paste the text into the value field. HDFView will only show the last line of the pasted text. However, after you copy and paste the text into another program you will see that HDFView stored the text including the newlines.



Figure 8: HDFView: Comment Attribute

2 Tutorials

2.1 Sensor Device Renaming

1. Please start IC0c:

```
icoc
```

2. The text based interface will show you a selection of the available devices:

```
IC0c
-----
Name      Address      RSSI
-----
1: Serial  08:6b:d7:01:de:81  -44 dBm
```

```
1-9: Connect to STH
```

```
f: Change Output File Name
```

```
n: Change STH Name
```

```
q: Quit IC0c
```

Choose the STH you want to rename by entering the number to the left of the device (here 1). To confirm your selection press the return key .

3. Now the menu should look like this:

```
          ICOc
STH "Serial" (08:6b:d7:01:de:81)
-----
```

```
Hardware Version      1.4.0
Firmware Version      2.1.10
Firmware Release Name Tanja
Serial Number         -
```

```
Supply Voltage        3.16 V
Chip Temperature      26.2 °C
```

```
Run Time              ∞ s
```

```
Prescaler             2
Acquisition Time      8
Oversampling Rate     64
  Sampling Rate       9524
Reference Voltage      VDD
```

```
Sensors               X
```

```
s: Start Data Acquisition
```

```
n: Change STH Name
```

```
r: Change Run Time
```

```
a: Configure ADC
```

```
p: Configure Sensors
```

```
0: Set Standby Mode
```

```
q: Disconnect from STH
```

Press the button **n** to change the name.

4. Enter the new device name.

```
New STH name (max. 8 characters): Blubb
```

Confirm the name with the return key **.**

5. The interface should now show you the menu of step 3. To disconnect from the holder press **e**.
6. Now you see the main menu of ICOc. The STH will show up under the name you used in step 4.
7. To exit ICOc, please use the key **q**.

2.2 Command Line Usage of ICOc

The ICOc program accepts optional command line arguments at startup. This way you can set default values for often used options. If you specify

- the name or
- Bluetooth address

of a sensor device, then you can even use ICOC without any user interaction, since in this case the program will immediately connect to the specified device and start the measurement process.

2.2.1 Available Options

To show the available command line options you can use the option `-h`:

```
icoc -h
```

which should show you the following output:

```
usage: icoc [-h] [-b BLUETOOTH_ADDRESS | -n NAME] [-f FILENAME] [-r SECONDS] [-1 FIRST_CHANNEL]
           [-2 SECOND_CHANNEL] [-3 THIRD_CHANNEL] [-s 2-127] [-a {1,2,3,4,8,16,32,64,128,256}]
           [-o {1,2,4,8,16,32,64,128,256,512,1024,2048,4096}]
           [-v {1V25,Vfs1V65,Vfs1V8,Vfs2V1,Vfs2V2,2V5,Vfs2V7,VDD,5V,6V6}]
           [--log {debug,info,warning,error,critical}]
```

Configure and measure data with the ICOTronic system

options:

`-h, --help` show this help message and exit

Connection:

`-b BLUETOOTH_ADDRESS, --bluetooth-address BLUETOOTH_ADDRESS`
connect to device with specified Bluetooth address (e.g. "08:6b:d7:01:de:81")

`-n NAME, --name NAME` connect to device with specified name

Measurement:

`-f FILENAME, --filename FILENAME`
base name of the output file (default: Measurement)

`-r SECONDS, --run-time SECONDS`
run time in seconds (values equal or below "0" specify infinite runtime)
(default: 0)

`-1 FIRST_CHANNEL, --first-channel FIRST_CHANNEL`
sensor channel number for first measurement channel (1 - 255; 0 to disable)
(default: 1)

`-2 SECOND_CHANNEL, --second-channel SECOND_CHANNEL`
sensor channel number for second measurement channel (1 - 255; 0 to disable)
(default: 0)

`-3 THIRD_CHANNEL, --third-channel THIRD_CHANNEL`
sensor channel number for third measurement channel (1 - 255; 0 to disable)
(default: 0)

ADC:

`-s 2-127, --prescaler 2-127`
Prescaler value (default: 2)

`-a {1,2,3,4,8,16,32,64,128,256}, --acquisition {1,2,3,4,8,16,32,64,128,256}`
Acquisition time value (default: 8)

`-o {1,2,4,8,16,32,64,128,256,512,1024,2048,4096}, --oversampling {1,2,4,8,16,32,64,128,256,512,1024,2048,4096}`
Oversampling rate value (default: 64)

```
-v {1V25,Vfs1V65,Vfs1V8,Vfs2V1,Vfs2V2,2V5,Vfs2V7,VDD,5V,6V6}, --voltage-reference {1V25,Vfs1V65,Vfs1V8,Vfs2V1,Vfs2V2,2V5,Vfs2V7,VDD,5V,6V6}
Reference voltage (default: VDD)
```

Logging:

```
--log {debug,info,warning,error,critical}
Minimum level of messages written to log (default: warning)
```

All options below the section “Measurement” and “ADC” in the help output of ICOC allow you to change the value of a specific configuration values before you start ICOC.

2.2.2 Channel Selection

To enable the measurement for the first (“x”) channel and second (“y”) channel of an “older” STH (Firmware 2.x, BGM113 chip) you can use the following command:

```
icoc -1 1 -2 1 -3 0
```

Here 0 indicates that you want to disable the channel while a positive number (such as 1) specifies that the measurement for the channel should take place. Since the default value

- for the option -1 is already 1, and
- for the option -3 is already 0

you can also leave out these options to arrive at the shorter command:

```
icoc -2 1
```

Note: Due to a problem in the current firmware the amount of **paket loss is much higher**, if you

- use the standard ADC configuration values, and
- enable data transmission for **exactly 2 (channels)**.

We strongly recommend you **use either one or three channels**.

For newer STH versions (Firmware 3.x, BGM121 chip) or SMHs (Sensory Milling Heads) you can also change the hardware/sensor channel for the first, second and third measurement channel. For example, to select

- hardware channel 8 for the first measurement channel
- hardware channel 1 for the second measurement channel, and
- hardware channel 3 for the third measurement channel

you can use the following command:

```
icoc -1 8 -2 1 -3 3
```

Note: If you connect to an older STH using the command above, then the command would just enable the measurement for all three measurement channels, but not change the selected hardware channel.

If you just want to enable/set a measurement channel and use the hardware channel with the same number you can also just leave the argument for the specific measurement channel empty. For example, to use

- hardware channel 1 for measurement channel 1,
- hardware channel 2 for measurement channel 2, and
- hardware channel 3 for measurement channel 3

you can use the following command:

```
icoc -1 -2 -3
```

or even shorter, since the default value for measurement channel 1 is hardware channel 1:

```
icoc -2 -3
```

2.2.3 Changing the Run Time

To change the run time of the measurement you can use the option `-r`, which takes the runtime in seconds as argument. The command

```
icoc -r 300
```

for example, would change the runtime to 5 minutes ($5 \cdot 60$ seconds = 300 seconds).

2.2.4 Start the Measurement

If you specify one of the options

- `-b/--bluetooth-address` or
- `-n/--name`

then ICOC will try to connect immediately to the specified device and start the measurement run. For example, to acquire acceleration data from the device with the (Bluetooth advertisement) **name “Blubb”** you can use the following command:

```
icoc -n Blubb
```

To read acceleration values for **5 seconds** from the device with the **Bluetooth address 08:6b:d7:01:de:81** you can use the following command:

```
icoc -b 08:6b:d7:01:de:81 -r 5
```

2.2.5 Changing the Logging Level

By default ICOC only writes log messages of level **ERROR** or higher. In some situations, for example when ICOC behaves incorrectly, you might want to set a lower level. You can do that using the option `--log`. For example, to activate logging of level **WARNING** and higher when you start ICOC you can use the following command:

```
icoc --log warning
```

The different logs for ICOC are stored in the root directory of the repository in the following files:

- `cli.log`: Log messages of ICOC
- `network.log`: Log messages of CAN network class
- `plotter.log`: Log messages of plotter (window process)

2.2.6 Changing the Reference Voltage

For certain sensor devices you have to change the reference voltage to retrieve a proper measurement value. For example, STHs that use a ± 40 g acceleration sensor (ADXL356) require a reference voltage of 1.8 V instead of the usual supply voltage (VDD) of 3.3 V. To select the correct reference voltage for these devices at startup use the option `-v Vfs1V8`:

```
icoc -v Vfs1V8
```

2.3 Production Tests

This tutorial lists the usual steps to test a sensory holder assembly or a sensory tool holder.

2.3.1 General

To run the production tests for one of the ICOTronic devices, please execute one of the following commands:

Device	Command
Stationary Transceiver Unit (STU)	<code>test-stu</code>
Sensory Holder Assembly (SHA), Sensory Tool Holder (STH)	<code>test-sth</code>
Sensory Milling Head (SMH)	<code>test-smh</code>

For a list of available command line options, please use the option `-h` after one of the commands e.g.:

```
test-sth -h
```

2.3.1.1 Specific Tests To only run a single test you need to specify its name. For example, to run the test `test__firmware_flash` of the STU you can use the following command:

```
test-stu TestSTU.test__firmware_flash
```

You can also run specific tests using pattern matching. To do that use the command line option `-k`. For example to run the firmware flash and the connection test of the STH test you can use the command:

```
test-sth -k flash -k connection
```

which executes all tests that contain the text `flash` or `connection`.

2.3.2 STH

The text below gives you a more detailed step-by-step guide on how to run the tests of the STH.

1. **Note:** You can **skip this step, if you do not want to run the flash test**. To skip the flash test, please set `STH` \rightarrow `STATUS` in the configuration file to `Epoxied`.

Please either

- create a directory called `STH`, or
- clone the STH repository to a location

beside this repository inside your file system. Then create a directory called **builds** and put the current version of the STH firmware into this directory. Afterwards the directory and file structure should look like this.

```
.
├── IC0c
│   └── STH
│       └── builds
│           └── manufacturingImageSthv2.1.10.hex
```

As alternative to the steps above you can also change the variable **STH** → **Firmware** → **Location** → **Flash** in the configuration file to point to the firmware that should be used for the flash test.

2. Make sure that the configuration value in the config file are set correctly. You probably need to change at least the following variables:
 - **Name:** Please change the Bluetooth advertisement name (**STH** → **NAME**) to the name of the STH you want to test.
 - **Serial Number of Programming Board:** Please make sure, that the variable **STH** → **PROGRAMMING BOARD** → **SERIAL NUMBER** contains the serial number of the programming board connected to the STH. This serial number should be displayed on the bottom right of the LCD on the programming board.
3. Now please open your favorite Terminal application and execute, the STH test using the command **test-sth**. For more information about this command, please take a look at the section “General” above.

Please note, that the test will rename the tested STH

- to a **Base64 encoded version of the Bluetooth MAC address**, if **STH** → **STATUS** is set to **Bare PCB**, or
- to the **serial number** (**STH** → **PROGRAMMING BOARD** → **SERIAL NUMBER**), if you set the status to **Epoxied**.

2.3.3 SMH

The preparation steps for the SMH test are very similar to the ones of the STH test.

1. Please make sure that the config value that stores the SMH firmware filepath (**SMH** → **Firmware** → **Location** → **Flash**) points to the correct firmware. If you have not downloaded a firmware image for the SMH you can do so here.
2. Check that the configuration values like SMH name (**SMH** → **NAME**) and programming board serial number (**SMH** → **PROGRAMMING BOARD** → **SERIAL NUMBER**) are set correctly.
3. Please execute the test using the following command:

```
test-smh
```

2.3.4 STU

The following description shows you how to run the STU tests.

1. **Note:** You can **skip this step, if you do not want to run the flash test.**

Please take a look at step 1 for the STH and replace every occurrence of STU with STU. In the end of this step the directory structure should look like something like this:

```
.
├── IC0c
├── STU
│   └── builds
│       └── manufacturingImageStuv2.1.10.hex
```

You can find the current version of the STU firmware [here](#).

2. Please take a look at the section “General” to find out how to execute the production tests for the STU. If you want to run the connection and EEPROM test (aka **all tests except the flash test**), then please execute the following command:

```
test-stu -k eeprom -k connection
```

Note: For the STU (flash) test to work correctly please connect the **programming board** to the USB port of the computer and the programming port of the STU **first**. Only after that connect the power injector to the power adapter. If you reverse this order, then the programmer might not work. If you do not connect the power injector, then the STU test might fail because of a CAN bus error:

Bus error: an error counter reached the ‘heavy’/‘warning’ limit

2.3.5 Firmware Versions

The (non-exhaustive) table below shows the compatible firmware for a certain device. The production tests assume that you use **firmware that includes the bootloader**.

Device	Hardware Version	Microcontroller	Firmware
STH	1.3	BGM113	• Version 2.1.10
STH	2.2	BGM123	• Aladdin
SMH	2.1	BGM121	• Version 3.0.0 • Version E3016 Beta

2.4 Verification Tests

2.4.1 Preparation

- The tests assume that the name of the STH is stored in `STH → NAME` in the configuration file `config.yaml`.
- Some of the STH tests assume that you connected the SHA/STH or STU via the programming cable. Please do that, since otherwise these tests will fail.

2.4.2 Execution

To run the verification tests for the STH, please enter the following command:

```
test-sth-verification -v
```

To execute the STU verification tests, you can use the command:

```
test-stu-verification -v
```

Please note that while most of the tests should run successfully, if you use working hardware and firmware, some of them might fail occasionally. In this case please rerun the specific test using the option `-k` and specifying a text that matches the name of the test. For example to return the STH test `test0107BlueToothConnectMin` you can use the following command:

```
test-sth-verification -v -k test0107
```

2.4.3 Problematic Tests

The tables below contains a list of tests that failed using a working SHA/STH and STU before. It should provide you with a good overview of which of the verification tests might fail, even if the hardware and firmware works correctly.

2.4.3.1 STH

Date	Failed Tests
2021-09-29	• test0107BlueToothConnectMin • test0332SignalIndicatorsAccZ • test0334SignalIndicatorsMulti • test0345MixedStreamingAccYZVoltBat
2021-09-30	• test0332SignalIndicatorsAccZ • test0334SignalIndicatorsMulti • test0345MixedStreamingAccYZVoltBat
2021-09-30	• test0332SignalIndicatorsAccZ • test0334SignalIndicatorsMulti • test0532MessageCountersAccZBattery
2021-10-05	• test0334SignalIndicatorsMulti • test0347StreamingAccXSingleBattery
2021-10-06	• test0109BlueToothRssi • test0334SignalIndicatorsMulti
2021-10-06	• test0107BlueToothConnectMin • test0334SignalIndicatorsMulti • test0345MixedStreamingAccYZVoltBat
2021-10-07	• test0107BlueToothConnectMin • test0332SignalIndicatorsAccZ • test0334SignalIndicatorsMulti • test0344MixedStreamingAccXYVoltBat • test0345MixedStreamingAccYZVoltBat
2021-10-11	• test0015PowerConsumptionEnergySaveMode2 • test0016PowerConsumptionEnergySaveModeAdv4000ms • test0332SignalIndicatorsAccZ • test0334SignalIndicatorsMulti • test0508AdcConfigSingle

Date	Failed Tests
2021-10-12	• test0332SignalIndicatorsAccZ • test0334SignalIndicatorsMulti • test0532MessageCountersAccZBattery
2021-10-13	• test0332SignalIndicatorsAccZ • test0334SignalIndicatorsMulti • test0508AdcConfigSingle • test0509AdcConfigDouble • test0510AdcConfigTripple • test0525MessageCounterAccZ
2021-12-09	• test0107BlueToothConnectMin • test0510AdcConfigTripple • test0523MessageCounterAccX • test0527MessageCounterAccXZ • test0529MessageCounterAccXYZ
2021-12-14	• test0107BlueToothConnectMin
2022-05-17	• test0015PowerConsumptionEnergySaveMode2 • test0016PowerConsumptionEnergySaveModeAdv4000ms • test0334SignalIndicatorsMulti • test0344MixedStreamingAccXYVoltBat • test0357StreamingAccXYSingleBattery

2.4.3.2 STU

Date	Failed Tests
2021-12-14	• test0102BlueToothConnectDisconnectDevice

2.5 Virtualization

You can also use (parts of) ICOC with various virtualization software. For that to work you have to make sure that (at least) the PEAK CAN adapter is attached to the virtual guest operating system. For some virtualization software you might have to install additional software for that to work. For example, VirtualBox requires that you install the VirtualBox Extension Pack before you can use USB 2 and USB 3 devices.

Note: Please be advised that the **VirtualBox Extension Pack is paid software** even though you can download and use it without any license key. **Oracle might come after you, if you do not pay for the license**, even if you use the Extension Pack in an educational setting.

The table below shows some of the virtualization software we tried and that worked (when we tested it).

Virtualization Software	Host OS	Host Architecture	Guest OS	Guest Architecture
Parallels Desktop	macOS	x64	Ubuntu 20.04	x64
Parallels Desktop	macOS	x64	Windows 10	x64
Parallels Desktop	macOS	ARM64	Fedora 36	ARM64
VirtualBox	macOS	x64	Windows 10	x64
VirtualBox	Windows	x64	Fedora 36	x64
WSL 2	Windows	x64	Ubuntu 20.04	x64

2.5.1 Windows Subsystem for Linux 2

Using ICOC in the WSL 2 currently requires using a custom Linux kernel. We **would not recommend** using ICOC with this type of virtualization software, since the setup requires quite some amount of work and time. Nevertheless the steps below should show you how you can use the PEAK CAN adapter and hence ICOC with WSL 2.

1. Install WSL 2 (Windows Shell):

```
wsl --install
```

2. Install Ubuntu 20.4 VM (Windows Shell):

```
wsl --install --distribution ubuntu
wsl --set-version Ubuntu 2
wsl --setdefault Ubuntu
```

3. Create Custom Kernel

Windows Shell:

Note: Please replace `<user>` with your (Linux) username (e.g. `rene`)

```
cd ~/Documents
mkdir WSL
cd WSL
wsl --export Ubuntu CANbuntu.tar
wsl --import CANbuntu CANbuntu CANbuntu.tar
wsl --set-version CANbuntu 2
wsl --distribution CANbuntu --user <user>
```

Linux Shell:

```
sudo apt update
sudo apt upgrade
sudo apt install build-essential flex bison libssl-dev libelf-dev \
                libncurses-dev autoconf libudev-dev libtool dwarves
cd ~
git clone https://github.com/microsoft/WSL2-Linux-Kernel.git
cd WSL2-Linux-Kernel
uname -r # 5.10.102.1-microsoft-standard-WSL2 → branch ...5.10.y
git checkout linux-msft-wsl-5.10.y
cat /proc/config.gz | gunzip > .config
make menuconfig
```

Enable these additional features:

- Device Drivers → USB Support
- Device Drivers → USB Support → USB announce new devices
- Device Drivers → USB Support → USB Modem (CDC ACM) support
- Device Drivers → USB Support → USB/IP
- Device Drivers → USB Support → USB/IP → VHCI HCD
- Device Drivers → USB Serial Converter Support
- Device Drivers → USB Serial Converter Support → USB FTDI Single port Serial Driver
- Networking support → CAN subsystem support
- Networking support → CAN subsystem support → Raw CAN Protocol
- Networking support → CAN subsystem support → CAN device drivers → Virtual Local CAN Interface

- Networking support → CAN subsystem support → CAN device drivers → Serial / USB serial CAN Adaptors (slcan)
- Networking support → CAN subsystem support → CAN device drivers → → CAN USB Interfaces → PEAK PCAN-USB/USB Pro interfaces for CAN 2.0b/CAN-FD

Save the modified kernel configuration.

Linux Shell:

```
touch .scmversion
make
sudo make modules_install
sudo make install
```

4. Install usbipd-win (Linux Shell):

```
cd tools/usb/usbip
./autogen.sh
./configure
make install
sudo cp libsrc/.libs/libusbip.so.0 /lib/libusbip.so.0
sudo apt-get install hwdata
```

5. Copy image (Linux Shell):

Note: Please replace <user> with your (Windows) username (e.g. **rene**)

```
cd ~/WSL2-Linux-Kernel
cp arch/x86/boot/bzImage /mnt/c/Users/<user>/Documents/WSL/canbuntu-bzImage
```

6. Create .wslconfig in (root of) Windows user directory and store the following text:

```
[wsl2]
kernel=c:\\users\\<user>\\Documents\\WSL\\canbuntu-bzImage
```

Note: Please replace <user> with your (Windows) username (e.g. **rene**)

7. Set default distribution (Windows Shell)

```
wsl --setdefault CANbuntu
```

8. Shutdown and restart WSL (Windows Shell):

```
wsl --shutdown
wsl
```

9. Change default user of WSL distro (Linux Shell):

```
nano /etc/wsl.conf
```

Insert the following text:

```
[user]
default=<user>
```

Note: Please replace <user> with your (Windows) username (e.g. **rene**)

Save the file and exit **nano**:

1. Ctrl + O
- 2.
3. Ctrl + X)

10. Restart WSL: See step 9

11. Install **usbipd** (Windows Shell):

```
winget install usbipd
```

12. Attach CAN-Adapter to Linux VM (Windows Shell)

```
usbipd wsl list
# ...
# 2-2      0c72:0012  PCAN-USB FD                      Not attached
# ...
usbipd wsl attach --busid 2-2
usbipd wsl list
# ...
# 2-2      0c72:0012  PCAN-USB FD                      Attached - CANbuntu
# ...
```

13. Check for PEAK CAN adapter in Linux (Linux Shell):

```
dmesg | grep peak_usb
# ...
# peak_usb 1-1:1.0: PEAK-System PCAN-USB FD v1 fw v3.2.0 (1 channels)
# ...

lsusb
# ...
# Bus 001 Device 002: ID 0c72:0012 PEAK System PCAN-USB FD
# ...
```

14. Add virtual link for CAN device (Linux Shell)

```
sudo ip link set can0 type can bitrate 1000000
sudo ip link set can0 up
```

15. Install **pip** (Linux Shell):

```
apt install python3-pip
```

16. Install **ICOc** (Linux Shell)

```
cd ~
mkdir Documents
cd Documents
git clone https://github.com/MyToolIT/ICOc.git
cd ICoc
pip install -e .
```

17. Run a script to test that everything works as expected (Linux Shell)

```
test-sth -k connection
```

Note: You only need to repeat steps

- 12: attach the CAN adapter to the VM in Windows and
- 14: create the link for the CAN device in Linux

after you set up everything properly once.

2.5.1.1 Installing/Using Simplicity Commander

1. Download and unpack Simplicity Commander (Linux Shell)

```
mkdir -p ~/Downloads
cd ~/Downloads
wget https://www.silabs.com/documents/public/software/SimplicityCommander-Linux.zip
unzip SimplicityCommander-Linux.zip
cd SimplicityCommander-Linux
tar xf Commander_linux_*.tar.bz
mkdir -p ~/Applications
mv commander ~/Applications
```

2. Add commander binary to path (Linux Shell)

1. Open ~/.profile in nano:

```
nano ~/.profile
```

2. Add the following text at the bottom:

```
if [ -d "$HOME/Applications/commander" ] ; then
    PATH="$HOME/Applications/commander:$PATH"
fi
```

3. Save your changes

3. Install JLink (Linux Shell)

1. Download JLink (64-bit DEB Installer) into your Windows user Downloads folder
2. Copy the installer

```
mv /mnt/c/Users/<user>/Downloads/*JLink_Linux*.deb ~/Downloads
```

Note: Please replace <user> with your (Windows) username (e.g. **rene**)

3. Install JLink package

```
cd ~/Downloads
sudo dpkg -i JLink_Linux_*.deb
sudo apt-get -f install # if there were unresolved dependencies
```

4. Detach USB connector of programming adapter

5. Attach USB connector of programming adapter
6. Reload udev rules (Linux Shell)

```
sudo service udev restart
sudo udevadm control --reload
```

7. Connect programming adapter to Linux (Windows Shell)

```
usbipd wsl list
# ...
# 2-1    1366:0105  JLink CDC UART Port (COM3), J-Link driver Not attached
# ...
usbipd wsl attach --busid 2-1
```

8. Check if commander JLink connection works without using sudo (Linux Shell)

```
commander adapter dbgmode OUT --serialno <serialnumber>
# Setting debug mode to OUT...
# DONE
```

Note: Please replace <serialnumber> with the serial number of your programming board (e.g. 440069950)

2.5.1.2 Run Tests in WSL

1. Reload udev rules (Linux Shell):

```
sudo service udev restart
sudo udevadm control --reload
```

2. Connect programming/CAN adapter to Linux (Windows Shell):

```
usbipd wsl list
# ...
# 2-1    1366:0105  JLink CDC UART Port (COM3), J-Link driver Not attached
# 2-2    0c72:0012  PCAN-USB FD                               Not attached
# ...
usbipd wsl attach --busid 2-1
usbipd wsl attach --busid 2-2
```

3. Add virtual link for CAN device (Linux Shell):

```
sudo ip link set can0 type can bitrate 1000000
sudo ip link set can0 up
```

4. Run tests (Linux Shell):

```
make run
```

3 Scripts

After you installed the ICOC package various helper scripts are available:

- **clean-repo**: Remove log (*.txt), HDF (*.hdf5) and PDF files (*.pdf) from the repository root
- **convert-base64-mac**: Utility to convert a Base64 encoded 8 character text into a Bluetooth MAC address
- **convert-mac-base64**: Convert Bluetooth MAC address into a (Base64) encoded 8 character string
- **check-eeeprom**: Write a byte value into the cells of an EEPROM page and check how many of the values are read incorrectly after a reset
- **icoc**: Controller and measurement software for the ICOTronic system
- **test-smh**: Test code for the SMH
- **test-sth**: Test code for the STH/SHA
- **test-stu**: Test code for the STU

3.1 EEPROM Check

The script **check-eeeprom** connects to an STH using its MAC address. Afterwards it writes a given byte value (default: 10) into all the cells of an EEPROM page. It then resets the STH, connects again and shows the amount of incorrect EEPROM bytes. It repeats the last steps 5 times before it prints all values in the EEPROM page.

The command below shows how to execute the EEPROM check for the STH with MAC address 08:6b:d7:01:de:81:

```
check-eeeprom 08:6b:d7:01:de:81
```

You can specify the value that should be written into the EEPROM cells using the option **--value**:

```
check-eeeprom 08:6b:d7:01:de:81 --value 42
```

3.2 ICOC

The command **icoc** calls **ui.py**. All command line arguments to the script will be directly forwarded to **ui.py**. For example, to read acceleration data for 10 seconds from the STH with the (Bluetooth advertisement) name **CGvXAd6B**, you can use the following command:

```
icoc -n CGvXAd6B -r 10
```

3.3 MAC Address Conversion

The utility **convert-mac-base64** returns the Base64 encoded version of a MAC address. We use the encoded addresses as unique Bluetooth advertisement name for the STH (or SHA). Unfortunately we can not use the MAC address directly because the maximum length of the name is limited to 8 characters. To decode the Base64 name back into a Bluetooth address you can use the script **convert-base64-mac**.

3.3.1 Examples

```
# Convert a MAC address into an 8 character name
convert-mac-base64 08:6b:d7:01:de:81
#> CGvXAd6B

# Convert the Base64 encoded name back into a MAC address
convert-base64-mac CGvXAd6B
#> 08:6b:d7:01:de:81
```

3.4 Remove Log, HDF and PDF Files

The script `clean-repo` removes

- HDF files (*.hdf5),
- PDF files (*.pdf), and
- log files (*.log)

from the repository root folder. To show the names of the removed files add the option `-v` (or `--verbose`):

```
clean-repo -v
```

If you also want to remove Python bytecode files from the repository we recommend you install `pyclean`:

```
pip install pyclean
```

and run the following commands in the root of the repository:

```
clean-repo -v; pyclean -v .
```

3.5 Test-STH

The command `test-sth` is a command that executes the tests for the STH (`sth.py`). All command line arguments of the wrapper will be forwarded to `sth.py`.

3.6 Test-STU

The command `test-stu` is a wrapper that executes the tests for the STU (`stu.py`). All command line arguments of the wrapper will be forwarded to `stu.py`.

3.7 Test-SMH

The command `test-smh` is a wrapper that executes the tests for the SMH (`smh.py`). All command line arguments of the wrapper will be forwarded to `smh.py`.

4 Development

4.1 Style

Please use the guidelines from PEP 8. For code formatting we currently use YAPF, which should format code according to PEP 8 by default.

To format the whole code base you can use the following command in the root of the repository:

```
yapf --in-place --parallel --recursive .
```

YAPF will not

- split long strings and
- add newlines to long lines not surrounded by parentheses.

To make sure that code has a maximum line length of 79 characters please split up long strings and add parentheses yourself.

For development we recommend that you use a tool or plugin that reformats your code with YAPF every time you save. This way we can ensure that we use a consistent style for the whole code base.

4.2 Tests

The following text describes some of the measures we should take to keep the software stable.

Please only push your changes to the **master** branch, if you think there are no new bugs or regressions. The **master** branch **should always contain a working version of the software**. Please **always run the automatic and manual tests** described below before you push to the **master** branch.

4.2.1 Code Checks

4.2.1.1 Flake8 We check the code with flake8:

```
pip install flake8
```

Please use the following command in the root of the repository to make sure you did not add any code that introduces warnings:

```
flake8
```

4.2.1.2 Mypy To check the type hint in the code base we use the static code checker Mypy:

```
pip install mypy
```

Please use the following command in the root of the repository to check the code base for type problems:

```
mypy mytoolit
```


4.2.2 Automatic Tests

4.2.2.1 Requirements Please install the pytest testing module:

```
pip install pytest
```

4.2.2.1.1 Usage Please run the following command in the root of the repository:

```
pytest -v
```

and make sure that it reports no test failures.

4.2.3 Manual Tests

4.2.3.1 ICOC

1. Call the command `icoc`.
2. Connect to a working STH (Enter the number and press `<enter>`)
3. Start the data acquisition (s)
4. After some time a window displaying the current acceleration of the STH (or SHA) should show up
5. Shake the STH
6. Make sure the window shows the increased acceleration
7. Close the window
8. The program should now exit, without any error messages

4.2.3.2 STH Test

1. Call the command `test-sth` for a working STH
2. Wait for the command execution
3. Check that the command shows no error messages
4. Open the PDF report (`STH_Test.pdf`) in the repository root and make sure that it includes the correct test data

4.2.3.3 STU Test

1. Call the command `test-stu` (or `test-stu -k eeprom -k connection` when you want to skip the flash test) for a working STU
2. Wait for the command execution
3. Check that the command shows no error messages
4. Open the PDF report (`STU_Test.pdf`) in the repository root and make sure that it includes the correct test data

4.2.3.3.1 Extended Tests The text below specifies extended manual test that should be executed before we release a new version of ICOC. Please note that the tests assume that you use more or less the default configuration values.

Check Command Line Interface

1. Open your favorite terminal application and change your working directory to the root of the repository
2. Remove log and data files from the repository:

```
clean-repo
```

3. Check that no HDF5 files exist in the repository. The following command should not produce any output:

```
ls *.hdf5
```

4. Give your test STH the name “Test-STH”
5. Measure data for 10 seconds using the following command:

```
icoc -n 'Test-STH' -r 10
```

6. Check that the repo now contains a HDF5 (*.hdf5) file

```
ls *.hdf5
```

7. Open the file in HDFView
8. Check that the table **acceleration** contains about 95 000 values
9. Check that the table contains three columns
10. Check that the meta attributes **Sensor_Range** and **Start_Time** exist
11. Check that **Sensor_Range** contains the correct maximum acceleration values for “Test-STH”
12. Check that **Start_Time** contains (roughly) the date and time when you executed the command from step 5
13. Check that ICOC handles the following incorrect program calls. The program should **not crash** and print a (helpful) **error description** (not a stack trace) before it exits.

```
icoc -b '12-12-12-12-12'  
icoc -n 'Toooolong'  
icoc -s 1  
icoc -a 257  
icoc -o -1  
icoc -1 ' -1'  
icoc -2 256  
icoc -3 nine
```

Check User Interface

1. Repeat steps 1. – 4. from the test above
2. Open ICOC using the following command:

```
icoc
```

3. The main menu of ICOC should show up
4. Try to connect to a non-existent STH
 1. Enter the text “1234”
 2. Press

3. ICOC should ignore the incorrect input and just display the main window
5. Change the output file name to “Test”
 1. Press f
 2. Remove the default name and enter the text “Test”
 3. Press
 4. After two seconds ICOC should show the main menu again
6. Connect to your test STH/SHA
 1. Enter the number besides “Test-STH”: Usually this will be the number “1”
 2. Press
 3. You should now be in the STH menu
7. Change the runtime to 20 seconds
 2. Press r
 3. Enter the text “hello”
 4. The last step should not have changed the default runtime of “0”
 5. Remove the default runtime (press)
 6. Enter the text “20”
 7. Press
8. Enable the first and second measurement channel
 1. Press p
 2. Remove the default axis config for the first measurement channel (press at least one time)
 3. Enter the characters “23456789ab”
 4. The last step should not have changed the empty input value
 5. Enable the first measurement channel:
 1. Press 1
 2. Press
 6. Enable the second measurement channel:
 1. Press
 2. Press 1
 3. Press
 7. Disable the third measurement channel:
 1. Press
 2. Press 0
 3. Press
9. Start the data acquisition
 1. Press s
 2. Shake the STH
 3. Make sure that shaking the STH changes (at least) the displayed value for the first measurement channel
 4. Wait until the measurement took place
10. Check the output file
 1. Check that the HDF5 output file exists: The filename should start with the characters “Test” followed by a timestamp and the extension “.hdf5”
 2. Open the HDF measurement file in HDFView
 3. Check that the table contains four columns
 4. One of the columns should have the name x
 5. Another column should have the name y

4.2.4 Combined Checks & Tests

While you need to execute some test for ICOC manually, other tests and checks can be automated.

Note: For the text below we assume that you installed `make` on your machine.

To run all checks, the STH test and the STU test use the following `make` command:

```
make run
```

Afterwards make sure there were no (unexpected) errors in the output of the STH and STU test.

4.3 Release

1. Make sure that **none** of the tests fail

- **Note:** Please execute `test-sth`

1. once with **STH** → **Status** set to **Epoxied**, and
2. once set to **Bare PCB**

in the configuration. To make sure, that the STU flash test also works, please use both STU test commands described in the section “STU Test”.

If you follow the steps above you make sure that the **flash tests work** for both STU and STH, and there are **no unintentional consequences of (not) flashing the chip** before you run the other parts of the test suite.

2. Execute the extended manual tests and check that everything works as expected

3. Create a new release here

1. Open the release notes for the latest version

2. Replace links with a permanent version:

For example instead of

- `../..something.txt` use
- `https://github.com/MyToolit/ICOC/blob/REVISION/something.txt`,

where **REVISION** is the latest version of the master branch (e.g. 8568893f for version 1.0.5)

3. Commit your changes

4. Copy the release notes

5. Paste them into the main text of the release web page

6. Decrease the header level of each section by two

7. Remove the very first header

8. Check that all links work correctly

4. Change the `__version__` number inside the `mytoolit` package

5. Push the latest two commits

6. Insert the version number (e.g. 1.0.5) into the tag field

7. For the release title use “Version **VERSION**”, where **VERSION** specifies the version number (e.g. “Version 1.0.5”)

8. Click on “Publish Release”

5 Releases

5.1 Version 1.6.0

5.1.1 ICOC

5.1.2 Internal

5.1.2.1 Network

- Removed the coroutine `read_x_acceleration` (please use `read_streaming_data_single`) instead
- Added the coroutines

Name	Description
<code>read_streaming_data_seconds</code>	Read streaming data for certain amount of time
<code>read_streaming_data_amount</code>	Read certain amount of streaming values

5.2 Version 1.5.0

5.2.1 ICOC

5.2.1.1 Command Line

- You can now set/enable the measurement for channel 2 and 3 just by providing the option for the measurement channel without an argument. In this case the measurement channel will be mapped to the corresponding hardware/sensor channel, i.e.
 - measurement channel 2 will use hardware channel 2 and
 - measurement channel 3 will use hardware channel 3.

Example:

```
icoc -2
```

The command above will use

- hardware channel 1 for measurement channel 1 (default value),
 - hardware channel 2 for measurement channel 2, and
 - disable measurement channel 3.
- ICOC can now connect to an STH, which has “no name” using the command line option `-n` (`--name`):

```
icoc -n '' # alternatively you can also just use `icoc -n`
```

5.2.1.2 Logging

- The log level set in the command line interface is now correctly propagated to the CAN class.

5.2.2 STH Tests

- The STH test script now uses the hardware version in the configuration (STH → `HARDWARE VERSION`) to determine the correct chip for the flash process.

Hardware Version	Chip
1.x.x	BGM113A256V2
2.x.x	BGM123A256V2

- The test that executes self test now only checks that the absolute difference between the voltage
 - at the the self test and
 - before/after the self test

is larger than a certain value. Before we always assumed that the voltage at the test is higher than before/after. This does not seem to be the case for certain sensors like the ± 40 g sensor ADXL356.

- The EEPROM test now also write the
 - acceleration slope and
 - acceleration offset values of the y-axis and z-axis into the EEPROM

5.2.3 Internal

5.2.3.1 Network

- Added the following coroutines:

Method	Description
<code>write_sensor_configuration</code>	Write measurement channel configuration
<code>read_eeprom_y_axis_acceleration_slope</code>	Read y-axis acceleration slope
<code>write_eeprom_y_axis_acceleration_slope</code>	Write y-axis acceleration slope
<code>read_eeprom_y_axis_acceleration_offset</code>	Read y-axis offset slope
<code>write_eeprom_y_axis_acceleration_offset</code>	Write y-axis offset slope
<code>read_eeprom_z_axis_acceleration_slope</code>	Read z-axis acceleration slope
<code>write_eeprom_z_axis_acceleration_slope</code>	Write z-axis acceleration slope
<code>read_eeprom_z_axis_acceleration_offset</code>	Read z-axis offset slope
<code>write_eeprom_z_axis_acceleration_offset</code>	Write z-axis offset slope

- The doctests of the Network class should now work more reliable on Linux

5.3 Version 1.4.0

5.3.1 ICOc

5.3.1.1 Channel Selection

- ICOc now supports more sensor channels. You can now select one of up to 255 (instead of 8) hardware channels for each of the three measurement channels.
- The CLI interface now uses three options (`-1`, `-2`, `-3`), instead of one option (`-p`), to specify the channel number of each of the three measurement channels.

5.3.1.2 Logging We replaced our custom logging class in the ICOC command line interface with logging code from the Python standard library. This logging code stores data in the following files:

- `cli.log`: Log messages of ICOC
- `network.log`: Log messages of CAN network class
- `plotter.log`: Log messages of plotter (window process)

By default ICOC only write log messages of level `ERROR` or higher into these files. To print more detailed output you can use the option `--log` to change the log level. For example, to change the log level to `DEBUG` (or higher) you can use the following command:

```
icoc --log debug
```

The other parts of the code, which use the old `Network` class (e.g. the verification test) do not store the log output in files, but instead use the standard error output (`stdout`). This change should hopefully improve the visibility of important log messages.

5.3.2 Tests

- The production tests for the hardware (STH, SMH and STU) should now also work on macOS, after you installed the PCUSB library.
- The STH tests now use the correct ADC reference voltage for the ± 40 g acceleration sensor ADXL356.

5.3.3 Internal

5.3.3.1 Measurement

- Renamed the method `convert_voltage_adc_to_volts` to `convert_to_supply_voltage`
- The method `convert_to_supply_voltage` now supports different reference voltages

5.3.3.2 Network

- The methods to
 - activate (`activate_acceleration_self_test`) and
 - deactivate (`deactivate_acceleration_self_test`)

the self test of the accelerometer now have a parameter to specify the dimension (x, y or z) for the self test.

- The method to read the acceleration voltage (`read_acceleration_voltage`) now supports two additional parameters to
 - specify the dimension (x, y or z) and
 - the ADC reference voltage.
- The method to read the supply voltage (`read_supply_voltage`) now takes the current reference voltage into consideration
- The new method `read_sensor_configuration` can be used to read the current sensor configuration (i.e. the mapping from hardware/sensor channel to measurement channel).

5.4 Version 1.3.0

5.4.1 ICOc

- The program can now change the number of the measured sensor channels. Please note:
 - This only works with the **latest version of sensor hardware and firmware**
 - The channel config support is in the **very early stages of development**; For example, ICOc currently still assumes that all sensors read acceleration values

5.4.2 STH Test

- All code of the (STH test) should now use the new network class.

5.4.3 SMH Test

- We added a flash upload test for the sensory milling head PCB. To only execute this part of the SMH test you can use the following command:

```
test-smh -k flash
```

5.4.4 Internal

5.4.4.1 ICOc

- The streaming code now uses CAN read events instead of polling.

5.4.4.2 Logging

- We do not log the CAN streaming messages (every CAN message after you chose “Start Data Acquisition” in ICOc) any more. The reason behind this change is that ICOc is currently not able to handle logging and writing the data into a HFD5 file at the same time on “slower” processors (such as Intel’s Core i5-5300 @ 2.3 GHz).

5.4.4.3 Network

- We added methods to read and write the sleep and advertisement time values of a sensor device using **System** (Bluetooth) commands
- We renamed the following methods:

Old Name	New Name
connect_sth	connect_sensor_device
get_sths	get_sensor_devices

5.5 Version 1.2.0

5.5.1 ICOc

- ICOc now assumes that sensor ranges reported by an STH below 1 (i.e. ± 0.5 g) are incorrect. In this case ICOc will assume that the range is ± 100 g instead.

5.5.2 STH Test

- The acceleration sensor self test (`test_acceleration_self_test`) now uses the new CAN class.

5.5.3 SMH Test

- We added a basic test for the new sensory milling hardware (SMH). To execute the test, please use the command

```
test-smh
```

Currently the test:

- checks if the STU is able to **connect** to the device,
- checks if the raw **ADC sensor values** are roughly equal to the expected values, and
- writes, reads and checks the **EEPROM** values of the SMH.

5.5.4 Internal

5.5.4.1 Calibration

- We added a class to create “Calibration Measurement” message data bytes

5.5.4.2 Message

- The textual representation of the message class now includes an extended description for
 - Bluetooth commands
 - “Calibration Measurement” commands , and
 - streaming commands.

5.5.4.3 Network We added methods

- to activate and deactivate the self test of the accelerometer
- to measure the acceleration voltage
- to read the advertisement time and sleep time of the reduced energy mode (mode 1)

5.6 Version 1.1.0

5.6.1 ICOc

5.6.1.1 Command Line Interface

- **Removed unused options** from the command line interface
- All command line arguments should now be **checked for validity** before starting ICOc
- You do not need to connect the PCAN interface to list the help message of the command line interface:

```
icoc -h
```

anymore

5.6.1.2 User Interface

- IC0c should now use considerably **less CPU power**.
- The window for a connected STH now also displays the **sensor range** of the acceleration sensor (in multiples of g)
- We **removed unused menu items** from the user interface
- The **menu** part of the interface now uses a **border** to distinguish itself from the rest of the interface:

```
1-9: Connect to STH
```

```
f: Change Output File Name
```

```
n: Change STH Name
```

```
q: Quit IC0c
```

- IC0c now provides **default values** for nearly all configuration inputs
- IC0c does **not crash** any more if your **terminal window is too small** to display the whole interface. The interface will look garbled if you resize the window to a size that does not fit the whole interface. However, it will look fine after you resize the window to a proper size afterwards, **as long as you do not make the window “really small”** (e.g. leave only two lines for IC0c).

5.6.1.3 Errors

- IC0c now prints the error messages at the top of the output. This should make it easier to check the reason of an error.

5.6.1.4 Output

- IC0c now stores **acceleration data in HDF5** format. For more information please take a look at the documentation.

5.6.1.5 Plotter

- The **plotter window now displays the acceleration values as multiples of g** (9.81 m/s^2).

5.6.1.6 Removed Functionality

- We **removed** a lot of the **unused, untested and unmaintained functionality** of IC0c:
 - XML configuration
 - Code to write and read EEPROM data using Excel files

5.6.2 Verification Test

- We enabled most parts of the STH verification test again
- The STH and STU verification tests now use the STH name from the configuration file `config.yaml` (STH → NAME).

5.7 Version 1.0.14

5.7.1 Logger

- ICOc now always extend the logging file name with the postfix `_error`, if there was a problem.
For example, if you use the default name `ICOc.txt`, then the logging file will be named
 - `ICOc_TIMESTAMP_error.txt` (e.g. `ICOc_2021-08-25_10-23-04_error.txt`) instead of
 - `ICOc_TIMESTAMP.txt` (e.g. `ICOc_2021-08-25_10-23-04.txt`)if there were any problems. The behavior of ICOc was similar before. However, the name of the error file could be chosen freely.
- You can now specify the default (base) name of the logging file in the configuration (`Logger` → `ICOC` → `FILENAME`).
- You can now specify the directory where ICOc stores acceleration data in the configuration (`Logger` → `ICOC` → `DIRECTORY`).

5.7.2 Verification Tests

We added the old code of the STH verification tests. You can now execute these tests using the command

```
test-sth-verification
```

For more information about these tests, please take a look at the section “Verification Tests” here.

5.7.3 Internal

5.7.3.1 Message

- The string representation of a message (`repr`) now includes additional information for
 - the “Bluetooth Write Energy Mode Reduced” command
 - the EEPROM command “Read Write Request Counter”
- The code for the string representation now handles incorrect device number values for the acknowledgment message of the `Get number of available devices` properly
- The method `acknowledge` does not ignore the value of the `error` parameter any more

5.8 Version 1.0.13

5.8.1 Documentation

- The repo now contains a bookdown project for the documentation. The latest version of the bookdown output (HTML, PDF, and EPUB) is available at GitHub (just click on the latest “run” and then on the link “ICOc Manual”) or in the Bitrix24 drive.

5.8.2 ICOc

- The data acquisition should now work more reliable, since we fixed
 - an CAN message overflow bug, and
 - a bug that resulted in a “blocked” acceleration data window.

5.8.3 Production Test

- The
 - stationary acceleration test (`test_acceleration_single_value`),
 - supply voltage test (`test_battery_voltage`),
 - connection test (`test_connection`), and
 - EEPROM test (`test_eeprom`)

now use the new network class instead of the old network class

5.8.4 Internal

5.8.4.1 Calibration

- Add class `CalibrationMeasurementFormat` to specify the data bytes of a calibration measurement command.

5.8.4.2 Measurement

- Add function `convert_voltage_adc_to_volts` to convert (2 byte) streaming voltage values to a supply voltage in volts

5.8.4.3 Message

- The string representation of a message (`repr`) now includes additional information for the Get/Set State block command

5.8.4.4 Network

- Add the coroutine `get_state` to retrieve information about the current state of a node
- Add the coroutine `read_voltage` to read the supply voltage of a connected STH
- Add the coroutine `read_x_acceleration` to read the x acceleration of a connected STH

5.8.4.5 Streaming Format

- New class `StreamingFormat` to specify the format of streaming data
- New class `StreamingFormatVoltage` to specify the format of voltage streaming data
- New class `StreamingFormatAcceleration` to specify the format of acceleration streaming data

5.8.4.6 Utility

- The new function `add_commander_path_to_environment` adds the path to Simplicity Commander (`commander`) to the `PATH` environment variable

5.9 Version 1.0.12

5.9.1 Config

- We now use the common term “version” instead of “revision” to specify the current state of the hardware. We therefore renamed
 - STH → HARDWARE REVISION to
 - STH → HARDWARE VERSION

in the configuration file.

5.9.2 IC Oc

- The menu for a connected STH now displays the correct STH name instead of the text “Tanja”.
- The main menu now uses “IC Oc” instead of “MyToolIt Terminal” as title

5.9.3 Internal

5.9.3.1 Network

- Add the coroutines
 - read_eeprom_advertisement_time_2,
 - read_eeprom_batch_number,
 - read_eeprom_firmware_version,
 - read_eeprom_gtin,
 - read_eeprom_hardware_version,
 - read_eeprom_oem_data,
 - read_eeprom_operating_time,
 - read_eeprom_power_off_cycles,
 - read_eeprom_power_on_cycles,
 - read_eeprom_product_name,
 - read_eeprom_production_date,
 - read_eeprom_release_name,
 - read_eeprom_serial_number,
 - read_eeprom_sleep_time_2,
 - read_eeprom_under_voltage_counter,
 - read_eeprom_watchdog_reset_counter,
 - read_eeprom_x_axis_acceleration_offset, and
 - read_eeprom_x_axis_acceleration_slope

to read specific values of the EEPROM

- Add the coroutines
 - write_eeprom_advertisement_time_2,
 - write_eeprom_batch_number,
 - write_eeprom_firmware_version,
 - write_eeprom_gtin,
 - write_eeprom_hardware_version,
 - write_eeprom_oem_data,
 - write_eeprom_operating_time,
 - write_eeprom_power_off_cycles,

- write_eeprom_power_on_cycles,
- write_eeprom_product_name,
- write_eeprom_production_date,
- write_eeprom_release_name,
- write_eeprom_serial_number,
- write_eeprom_sleep_time_2,
- write_eeprom_under_voltage_counter,
- write_eeprom_watchdog_reset_counter,
- write_eeprom_x_axis_acceleration_offset, and
- write_eeprom_x_axis_acceleration_slope

to change specific values in the EEPROM

- Add the following coroutines to read product data:

Name	Data Item
get_gtin	GTIN (Global Trade Identification Number)
get_hardware_version	Hardware Version Number
get_firmware_version	Firmware Version Number
get_firmware_release_name	Firmware Release Name
get_serial_number	Serial Number
get_product_name	Product Name
get_oem_data	OEM (Free Use) Data

5.9.3.2 Status

- Add wrapper class for the `Get/Set State` command

5.10 Version 1.0.11

5.10.1 Documentation

- Add STH renaming tutorial

5.10.2 EEPROM Checker

- The EEPROM checker (`check-eeprom`) now uses the new network class.

5.10.3 ICOC

- We fixed the sporadic crashes of the graphical plotter interface. You should not see messages about “WinError 10061” any more, when you try to read acceleration data with `icoc`.

5.10.4 STH Test

- The test now uses the serial number (STH → SERIAL NUMBER) as new name, if you set the status (STH → STATUS) to `Epoxied` in the configuration. If you use a different status, then the test will still use the Base64 encoded MAC address as new (Bluetooth advertisement) name.
- Remove wait time (of 2 seconds) after Bluetooth connection was established. In theory this should make the test execution quite a bit faster, without any adverse effects.

5.10.5 Internal

5.10.5.1 Message

- The string representation of a message (`repr`) now includes additional information for:
 - EEPROM commands, and
 - the Bluetooth commands:
 - * to request a connection,
 - * to check the connection status,
 - * to deactivate the Bluetooth connection,
 - * to retrieve the RSSI, and
 - * to retrieve the MAC address
 - * to connect to a device using a MAC address

5.10.5.2 Network (Old)

- Improve error message for disconnected CAN adapter

5.10.5.3 Network (New)

- The class now sends requests multiple times, if it does not receive an answer in a certain amount of time
- Improve error message for disconnected CAN adapter
- The class now logs sent and received messages on the CAN bus, if you change the logger level to `DEBUG` (`LOGGER→CAN→LEVEL` in the configuration)
- Renamed the following coroutines:

Old Name	New Name
<code>get_available_devices_bluetooth</code>	<code>get_available_devices</code>
<code>get_device_name_bluetooth</code>	<code>get_name</code>
<code>get_mac_address_bluetooth</code>	<code>get_mac_address</code>
<code>connect_device_number_bluetooth</code>	<code>connect_with_device_number</code>
<code>check_connection_device_bluetooth</code>	<code>is_connected</code>

- Add the following coroutines

Name	Description
<code>connect_with_mac_address</code>	Connect to a device using its MAC address
<code>get_rssi</code>	Retrieve the RSSI (Received Signal Strength Indication) of a device
<code>get_sths</code>	Retrieve a list of available STHs
<code>connect_sth</code>	Directly connect to an STH using its • MAC address, • device number, or • name
<code>set_name</code>	Set the (Bluetooth advertisement) name of an STU or STH

- Add coroutines:
 - `read_eeprom`,

- `read_eeprom_float`,
- `read_eeprom_int`, and
- `read_eeprom_text`

to read EEPROM data

- Add coroutines

- `write_eeprom`,
- `write_eeprom_float`,
- `write_eeprom_int`, and
- `write_eeprom_text`,

to write EEPROM data

- Add the coroutines:

- `read_eeprom_advertisement_time_1`,
- `read_eeprom_name`,
- `read_eeprom_sleep_time_1`, and
- `read_eeprom_status`

to read specific parts of the EEPROM

- Add the coroutines:

- `write_eeprom_advertisement_time_1`,
- `write_eeprom_name`,
- `write_eeprom_sleep_time_1`, and
- `write_eeprom_status`

to write specific parts of the EEPROM

5.11 Version 1.0.10

5.11.1 Checks

- We also check the code base with the static type checker Mypy

5.11.2 GitHub Actions

- We now also test ICOC on Ubuntu Linux

5.11.3 Package

- We now use the version number specified in the init file of the package for the package version number.
- You can now also install the Package on Linux (and macOS)

5.11.4 Requirements

- The package now uses the EUI (Extended Unique Identifier) class of the `netaddr` package to handle MAC addresses

5.11.5 Scripts

- We added a script that removes log and PDF files from the repository root. For more information please take a look at the section “Remove Log and PDF Files” of the script documentation.

5.11.5.1 STH Test

- The report of the STH test now also contains information about the used acceleration sensor (± 100 g or ± 50 g)
- We added a configuration item for the holder type (STH \rightarrow HOLDER TYPE)
- We added the holder type and the serial number to the PDF report
- Removed extra space before and after headers in PDF report
- The PDF report now contains a list of tables with checkboxes for manual checks

5.11.6 Internal

5.11.6.1 Checkbox

- New `Flowable` class that can be used to add a checkbox to a PDF report

5.11.6.2 Command

- Add method `is_error` to check if the current command represents an error
- Add method `set_error` to set or unset the error bit

5.11.6.3 Identifier

- Add method `acknowledge` to retrieve expected acknowledgment identifier for id
- Add method `is_error` to check if the current identifier represents an error message
- Add method `set_error` to set or unset the error bit
- Support comparison with other identifiers (`==`)

5.11.6.4 Message

- The class now uses the class `Message` of `python-can` to store data instead of the class `TPCANMsg` of the PCAN-Basic API
- Add method `identifier` to receive an identifier object for the current message
- Add accessor to set and retrieve message data (`.data`)
- The method `acknowledge` now stores the data of the message in the acknowledgment message
- Fix conversion into `python-can` message for non-empty data field
- Add explanation to string representation for Bluetooth “Activate” and “Get number of available devices” subcommand

5.11.6.5 Network (New)

- Add coroutine (`reset_node`) to reset a node in the network
- Add coroutine (`activate_bluetooth`) to activate Bluetooth on a node in the network
- Add coroutine (`get_available_devices_bluetooth`) to retrieve the number of available Bluetooth devices

- Add coroutine (`get_device_name_bluetooth`) to retrieve the Bluetooth advertisement name of a device
- Add coroutine (`connect_device_number_bluetooth`) to connect to a Bluetooth device using the device number
- Add coroutine (`deactivate_bluetooth`) to deactivate the Bluetooth connection of a node
- Add coroutine (`check_connection_device_bluetooth`) to check if a Bluetooth device is connected to a node
- Add coroutine (`get_mac_address_bluetooth`) to retrieve the MAC address of a connected Bluetooth device
- Implement context manager interface (`with ... as`)
- The code now checks part of the acknowledgement data for the Bluetooth and reset commands of the `System` block

5.11.6.6 Report

- Add method `add_checkbox_list` to add a list of checkboxes at the end of the PDF report

5.11.6.7 Utility

- Add function `bytearray_to_text` to convert byte data to a string

5.12 Version 1.0.9

5.12.1 Configuration

- We moved the configuration file for the test scripts into the `mytoolit` package.
- We moved the configuration file for ICOC into the `mytoolit` package.
- We now use uppercase letters for all configuration keys in `config.yaml`. The reason behind this update is that we can overwrite these values using environment variables – with the prefix `DYNACONF_` – even on Windows. Unfortunately Windows converts all environment variables to uppercase.

5.12.2 Compatibility

- This version of ICOC requires at least Python 3.7, since we use the `annotations` directive from the `__futures__` module

5.12.3 Package

- We added a package description for ICOC. You can now install the software using `pip install -e .` in the root of the repository. To uninstall the package use `pip uninstall icoc`.

5.12.4 Scripts

- We added a new EEPROM checking tool. For more information please take a look at the section “EEPROM Check” of the script documentation.

5.12.5 Internal

- We removed old hardware test code

5.12.5.1 Network (Old)

- Simplified code

5.13 Version 1.0.8

5.13.1 Internal

5.13.1.1 Message

- Added method to convert message to python-can message object
- Renamed initialization attribute `payload` to `data`
- Added support to initialize a message object with a message object of python-can

5.13.2 Production Test

- The name of the PDF test report now reflects the tested node. The latest test data for the STH will be stored in a file called `STH Test.pdf`, while the STU test data is stored in a file called `STU Test.pdf`. Before this update both tests would use the file name `Report.pdf`.

5.13.3 STH Test

- We now always assume the name of the STH (`STH → Name`) in the configuration is given as string. This improves the usability of the tests, since otherwise you might specify an integer as name (e.g. 1337) and wonder why the test is unable to connect to the STH.
- The STH test now prints a message about a possible incorrect config value for the acceleration sensor (`STH → Acceleration Sensor → Sensor`), if the self test of the accelerometer failed.
- The STH test fails, if you use an sensor value (`STH → Acceleration Sensor → Sensor`) that is not one of the supported values
 - ADXL1001 or
 - ADXL1002.

5.13.4 Tests

- We now check the code base with flake8.
- We use GitHub actions
 - to run non-hardware dependent parts of the automated tests, and
 - to check the code base with flake8

5.14 Version 1.0.7

5.14.1 Logging

- The main CAN class now logs the received CAN messages in a file called `can.log`, if you specify the level `DEBUG` in the configuration file.

5.14.2 STH Test

- The STH test case now skips the flash test if the status in the configuration file (STH → **Status**) is set to **Epoxied**.
- The test now also supports the ± 50 g digital accelerometer ADXL1002. To choose which sensor is part of the STH (or SHA), please change the value **STH → Acceleration Sensor → Sensor** to the appropriate value in the configuration file.
- Removed the over the air (OTA) test, since it requires the command `ota-dfu`, which needs to be compiled first, and often does not work reliable.
- The renaming of the STH in the EEPROM test should now work more reliable.

5.14.3 STU Test

- Add first version of new test for the STU. For more information, please take a look at the section “Production Tests” of the main readme.

5.15 Version 1.0.6

5.15.1 Documentation

- We added a tutorial that explains the necessary steps for releasing a new version of ICOC

5.16 Version 1.0.5

5.16.1 Configuration

- We moved the host and port configuration values of the Matplotlib GUI from the XML configuration file into the YAML configuration.

5.16.2 Documentation

- We documented how to execute the automatic tests for software.

5.16.3 STH Test

- The EEPROM tests now also writes the firmware release name (STH → **Firmware** → **Release Name** in the configuration) into the EEPROM.
- The EEPROM test now sets the
 - operating time,
 - power on cycles,
 - power off cycles,
 - under voltage counter, and the
 - watchdog reset counter

to 0.

5.17 Version 1.0.4

5.17.1 Documentation

- We moved the release notes from the init code of the `mytoolit` package into this file.
- We added an FAQ text that should answer questions not covered by the main ReadMe.
- We rewrote the main readme file. The document should now contain more information that is relevant for the typical user of ICOC. Before this update the text contained more technical information that is only interesting for developers of the ICOC application, such as ADC settings and hardware configuration steps.
- We added a tutorial that shows you how to test an STH (or SHA).
- The documentation now includes a description of manual tests for the software.
- We added a document that describes the style guidelines for the code base.

5.17.2 Install

- We forgot to add the requirements file for pip in the last release. This problem should now be fixed.
- We now use the Python CAN package to access the PCAN-Basic API instead of including a copy of the Python API file in the repository.

5.17.3 Scripts

- Add a simple wrapper script for the STH test. If you add its parent folder to your path you will be able to execute the test regardless of your current path. For more information, please take a look at the ReadMe.
- We added a simple wrapper script for `mwt.py`. For more information, please take a look here.
- The new scripts `Convert-MAC-Base64` and `Convert-Base64-MAC` convert a MAC address (e.g. `08:6b:d7:01:de:81`) into a Base64 encoded (8 character long) text (e.g. `CGvXAd6B`) and back. We use the Base64 encoded MAC address as Bluetooth advertisement name to uniquely identify a STH (or SHA).

5.17.4 Style

- We formatted the code base with YAPF.

5.17.5 STH Test

- We added a test that checks, if updating the over the air update via the `ota-dfu` command line application works correctly. Currently this test is not activated by default, since it requires that the operator compiles the `ota-dfu` application.
- The test now uses the Base64 encoded version of the MAC address as default name. The rationale behind this update was that the name we used until now was not unique for a certain STH (or SHA). For more information, please take a look here.
- The EEPROM test now resets the STH at the end. This way the STH will already use the Base64 encoded MAC address as name after the test was executed. Before this update we had to do a manual reset for the name change to take place.

5.18 Version 1.0.3

5.18.1 Bug Fixes

- The data acquisition should now work on additional machines

5.18.2 Compatibility

- Since the latest code base uses f-Strings the software now requires at least Python 3.6.

5.18.3 Requirements

- You can now install the required Python packages using the command

```
pip install -r requirements.txt
```

in the root of the repository.

5.18.4 STH Test

- The test prints its output to the

- standard and
- standard error output

instead of a log file. The program will still produce log files for the individual parts of the test, since the class that handles the CAN communication currently requires this behavior.

- The program now prints the various attributes of the tested STH, such as name and RSSI, on the standard output.
- The program stores the most important attributes of the STH and the result of the test in a PDF report (in the root of the repository).
- The various configuration data for the test is now stored in a single YAML file (`Configuration/config.yaml`).
- The test program is now location independent. This means you do not need to change the working directory to call the script any more. For example, executing the command:

```
py mytoolit/test/production/sth.py -k battery -k connect
```

or the commands:

```
cd mytoolit/test/production  
py sth.py -k battery -k connect
```

in the root of repository should have the same effect.

5.19 Version 1.0.2

- Bug Elimination

5.20 Version 1.0.0

Initial Release:

- Release Date: 6. September 2019
- MyToolIt Watch(MyToolItWatch.py): Supports MyToolIt Watch functionality
- MyToolItTerminal (mwt.py): Terminal access to MyToolIt functionality
- Data Base Functionality: configKeys.xml
- Access to EEPROM via Excel interface
- Uses CanFd.py module to access MyToolIt protocol
- Supports Tests for internal verification:
 - MyToolItTestSth.py
 - MyToolItTestSthManually.py
 - MyToolItTestStu.py
 - MyToolItTestStuManually.py
- Supports configuration files for tests:
 - SthLimits.py
 - StuLimits.py
 - testSignal.py
- Support graphical visualization in real time: Plotter.py