Remote Debugging

App über VSCode starten (ohne Installation)

1. Debug-Klasse ins Projekt einfügen und in Main importieren
2. Setup.py anpassen
    a. Debugpy unter „install_requirements“ einfügen
    b. Debug.py unter “scripts” einfügen

```
from setuptools import setup

setup(name = 'sdk-py-calculator',
      version='2.4.0',
      description = 'From setup.py: ctrlX Data Layer calculator that calculates operations',
      author = 'SDK Team',
      packages = ['alldataprovider', 'helper'],
      install_requires = ['ctrlx-datalayer', 'ctrlx-fbs', 'debugpy', 'setuptools'],
      scripts = ['main.py', 'debug.py'],
      license = 'MIT License'
)
```

3. Debugpy in requirements.txt einfügen
4. Debug-Funktion in Main-Entry-Point von main.py aufrufen
    a. debug.remote_debugging_wait_for_client(port=15678)
7. Verbindungseinstellungen im Datalayer-Helper anpassen

    (**/project/helper/ctrlx_datalayer_helper.py**)

```
ctrlx_datalayer_helper.py ×
helper > ctrlx_datalayer_helper.py > get_provider
 48
 49  def get_connection_string(
 50          ip="192.168.1.1",
 51          user="boschrexroth",
 52          password="boschrexroth",
 53          ssl_port=443):
 54 >    #region code ···
 75
 76
 77  def get_client(system: ctrlxdatalayer.system.System,
 78                 ip="192.168.1.1",
 79                 user="boschrexroth",
 80                 password="boschrexroth",
 81                 ssl_port=443):
 82 >    #region code ···
104
105  def get_provider(system: ctrlxdatalayer.system.System,
106                   ip="192.168.1.1",
107                   user="boschrexroth",
108                   password="boschrexroth",
109                   ssl_port=443):
110      #region code
```

8. F5 drücken bzw. Run -> Start Debugging

Build & Upload Snap

1. **build-upload-log-snap.sh** in App Build Environment einfügen (normalerweise im “Scripts”-Ordner in der SDK vorinstalliert)
    a. Gegebenenfalls den Pfad zum Skript in **tasks.json** anpassen

```
{} tasks.json ×
.vscode > {} tasks.json > [ ] tasks > {} 5
5   "tasks": [
41    {
42      "label": "Build upload snap - ctrlX COREvirtual Network Adapter",
43      "type": "shell",
44      "command": "../../scripts/build-upload-log-snap.sh",
45      "args": [
46        "-ctrlx-virt-NA"
47      ],
48      "problemMatcher": [],
49      "group": {
50        "kind": "build",
51        "isDefault": true
52      }
53    },{
54      "label": "Build upload snap - ctrlX CORE",
55      "type": "shell",
56      "command": "../../scripts/build-upload-log-snap.sh",
57      "args": [
58        "-build",
59        "${input:Build}",
60        "-upload",
61        "${input:Upload}",
62        "-logs",
63        "${input:Logs}",
64        "-svc",
65        "${input:Service}",
66        "-addr",
67        "${input:IPAddress}",
68        "-ssl-port",
69        "${input:SSLPort}",
70        "-ssl-usr",
71        "${input:SSLUser}",
72        "-ssl-pwd",
73        "${input:SSLPwd}",
74        "-arch",
75        "${input:Arch}",
76        "-ssh-port",
77        "${input:SSHPort}",
78        "-ssh-usr",
79        "${input:SSHUser}",
80      ],
81      "problemMatcher": [],
82      "group": {
83        "kind": "build",
84        "isDefault": true
85      }
86    },
```

    b. Gegebenenfalls Konfiguration im Skript anpassen

```
$ build-upload-log-snap.sh ×
$ build-upload-log-snap.sh

1  #!/bin/bash
2
3  ARCH=arm64
4
5  ADDR=192.168.1.1
6  SSL_PORT=443
7  SSH_PORT=22
8
9  ASK=y
10
11 SECONDS_TO_WAIT_AFTER_SHOW_ARGUMENTS=5
12 SECONDS_TO_WAIT_AFTER_UPLOAD=30
13 SECONDS_TO_WAIT_AFTER_UNINSTALL=30
14
15 BUILD=y
16 UPLOAD=y
17 UNINSTALL=n
18 BUNDLE=""
19
20 SSL_USR=boschrexroth
21 SSL_PWD=boschrexroth
22
23 SERVICE=y
24 OPERATING=y
25
26 SSH_USR=rexroot
27 SSH_PWD=rexroot
28
29 LOGS=y
```

2. Terminal -> Build Task -> z.B. “**upload snap – ctrlX CORE**”
    a. Alternativ auf dem CORE per SSH:
    sudo snap run sdk-py-remote-debug.app --debug-port=15678

    sudo snap stop sdk-py-remote-debug

[Ausführliche online Anleitung für CtrlX CORE 1.2]

SSH Aktivieren

1. SSH key generieren (App Build Environment)
    a. ssh-keygen -b 4096
2. SSH auf CORE aktivieren
3. SSH Key von Environment auf CORE kopieren (App Build Environment)
    a. Adapter/real CORE: **ssh-copy-id -i ~/.ssh/id_rsa.pub rexroot@192.168.1.1**
    b. Port Forwarding: **ssh-copy-id -i ~/.ssh/id_rsa.pub -p8022 rexroot@10.0.2.2**
4. Mit SSH verbinden
    a. ssh rexroot@192.168.1.1
    b. ssh -p8022 rexroot@10.0.2.2

Logging

Logs eines bestimmten Snap in der Konsole ausgeben

sudo snap logs -f sdk-py-logbook

```
rexroot@VirtualControl-9test:~$ sudo snap logs sdk-py-logbook
2024-06-26T06:03:00Z systemd[1]: Started Service for snap application sdk-py-logbook.logbook.
2024-06-26T06:03:05Z sdk-py-logbook.logbook[1117]: Simple snap in Python using logging with different log levels
2024-06-26T06:03:06Z root[1117]: I am an exception message
NoneType: None
2024-06-26T06:03:06Z root[1117]: I am a critical message
2024-06-26T06:03:06Z root[1117]: I am an error
2024-06-26T06:03:06Z root[1117]: I am a warning
2024-06-26T06:03:06Z root[1117]: I am an info message
2024-06-26T06:03:06Z root[1117]: I am a debug message
```

Logs eines bestimmten Services mit journalctl in der Konsole ausgeben

sudo journalctl --unit snap.sdk-py-logbook.logbook.service

```
rexroot@VirtualControl-9test:~$ sudo journalctl --unit snap.sdk-py-logbook.logbook.service
Jun 26 07:34:38 VirtualControl-9test systemd[1]: Started Service for snap application sdk-py-logbook.logbook.
Jun 26 07:34:41 VirtualControl-9test sdk-py-logbook.logbook[1129]: Simple snap in Python using logging with different log levels
Jun 26 07:34:41 VirtualControl-9test root[1129]: I am an exception message
                                                 NoneType: None
Jun 26 07:34:41 VirtualControl-9test root[1129]: I am a critical message
Jun 26 07:34:41 VirtualControl-9test root[1129]: I am an error
Jun 26 07:34:41 VirtualControl-9test root[1129]: I am a warning
Jun 26 07:34:41 VirtualControl-9test root[1129]: I am an info message
Jun 26 07:34:41 VirtualControl-9test root[1129]: I am a debug message
```

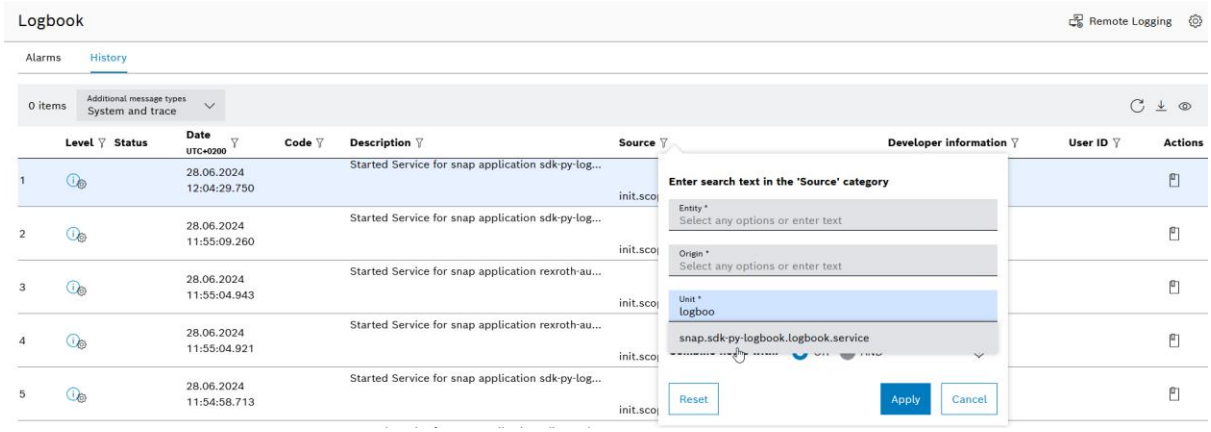
Logs einer bestimmten Kategorie, z.B.:

sudo journalctl -p err..alert

sudo journalctl -p debug

```
rexroot@VirtualControl-9test:~$ sudo journalctl -p err..alert
Jun 26 06:02:48 VirtualControl-9test kernel: ..MP-BIOS bug: 8254 timer not connected to IO-APIC
Jun 26 06:02:48 VirtualControl-9test kernel: [Firmware Bug]: the BIOS has corrupted hw-PMU resources (MSR 186 is 53003c)
Jun 26 06:02:48 VirtualControl-9test kernel: [Firmware Bug]: CPU1: APIC id mismatch. Firmware: 1 APIC: 0
Jun 26 06:02:48 VirtualControl-9test kernel: [Firmware Bug]: CPU2: APIC id mismatch. Firmware: 2 APIC: 0
Jun 26 06:02:48 VirtualControl-9test kernel: [Firmware Bug]: CPU3: APIC id mismatch. Firmware: 3 APIC: 0
Jun 26 06:03:04 VirtualControl-9test kernel: /dev/sda: Can't open blockdev
Jun 26 06:03:04 VirtualControl-9test kernel: /dev/sda3: Can't open blockdev
Jun 26 06:03:05 VirtualControl-9test kernel: /dev/sr0: Can't open blockdev
Jun 26 06:03:06 VirtualControl-9test root[1117]: I am an exception message
                                                 NoneType: None
Jun 26 06:03:06 VirtualControl-9test root[1117]: I am an error
```

Dieselben Einträge können auch per Webinterface in dem Logbook eingesehen werden. Dabei kann nach mehreren Möglichkeiten gefiltert werden. Meldungen einzelner Apps werden unter dessen Service angezeigt und ob eine App beispielsweise gestartet/beendet wurde, kann durch einen Filter der init.scope-Quelle eingesehen werden.



Auch das Level der Schweregrade kann gefiltert werden.