Aktor: Logger (CSVlog)

Dieser Aktor ist eine Contribution vom Makerspace Bocholt

```
mkdir -p /opt/fabinfra/adapters/csvlog/
```

```
cd /opt/fabinfra/adapters/csvlog/
python3 -m venv env
. env/bin/activate #activate venv

pip install paho-mqtt
```

```
vim /opt/fabinfra/adapters/csvlog/main.py
```

```
#!/usr/bin/env python3

import sys
import argparse
import paho.mqtt
import paho.mqtt.publish as publish
import paho.mqtt.client as mqtt
from csv import writer
import time

def append_list_as_row(file_name, list_of_elem):
    # Open file in append mode
    with open(file_name, 'a+', newline='') as write_obj:
        # Create a writer object from csv module
        csv_writer = writer(write_obj)
        # Add contents of list as last row in the csv file
        csv_writer.writerow(list_of_elem)



def on_free(args, actor_name):
    """
    Function called when the state of the connected machine changes to Free
    again
    """
    now = time.strftime('%d-%m-%Y %H:%M:%S')
    # List of strings
    row_contents = [now, actor_name, 'leer', 'released']
    # Append a list as new line to an old csv file
    append_list_as_row('log.csv', row_contents)

    print("logging event: release")
    if not args.quiet:
        print("process called with unexpected combo id %s and state 'Reserved'" % actor_name)
    exit(-1)

def on_use(args, actor_name, user_id):
    """
    Function called when an user takes control of the connected machine

    user_id contains the UID of the user now using the machine
    """
    now = time.strftime('%d-%m-%Y %H:%M:%S')
    # List of strings
    row_contents = [now, actor_name, user_id, 'booked']
    # Append a list as new line to an old csv file
    append_list_as_row('log.csv', row_contents)

    print("logging event: inUse")
    if not args.quiet:
        print("process called with unexpected combo id %s and state 'Reserved'" % actor_name)
    exit(-1)

def on_tocheck(args, actor_name, user_id):
    """
    Function called when an user returns control and the connected machine is
    configured to go to state `ToCheck` instead of `Free` in that case.

    user_id contains the UID of the manager expected to check the machine.
    The user that used the machine beforehand has to be taken from the last
    user field using the API (via e.g. the mobile app)
    """
    now = time.strftime('%d-%m-%Y %H:%M:%S')
    # List of strings
    row_contents = [now, actor_name, user_id, 'tocheck']
    # Append a list as new line to an old csv file
    append_list_as_row('log.csv', row_contents)

    print("logging event: toCheck")
    if not args.quiet:
        print("process called with unexpected combo id %s and state 'Reserved'" % actor_name)
    exit(-1)

def on_blocked(args, actor_name, user_id):
    """
    Function called when an manager marks the connected machine as `Blocked`

    user_id contains the UID of the manager that blocked the machine
    """



    now = time.strftime('%d-%m-%Y %H:%M:%S')
    # List of strings
    row_contents = [now, actor_name, user_id, 'blocked']
    # Append a list as new line to an old csv file
    append_list_as_row('log.csv', row_contents)

    print("logging event: blocked")
    if not args.quiet:
        print("process called with unexpected combo id %s and state 'Reserved'" % actor_name)
    exit(-1)



def on_disabled(args, actor_name):
    """
    Function called when the connected machine is marked `Disabled`
    """
    now = time.strftime('%d-%m-%Y %H:%M:%S')
    # List of strings
    row_contents = [now, actor_name, user_id, 'disabled']
    # Append a list as new line to an old csv file
    append_list_as_row('log.csv', row_contents)

    print("logging event: disabled")
    if not args.quiet:
        print("process called with unexpected combo id %s and state 'Reserved'" % actor_name)
    exit(-1)

def on_reserve(args, actor_name, user_id):
    """
    Function called when the connected machine has been reserved by somebody.

    user_id contains the UID of the reserving user.
    """
    now = time.strftime('%d-%m-%Y %H:%M:%S')
    # List of strings
    row_contents = [now, actor_name, user_id, 'reserved']
    # Append a list as new line to an old csv file
    append_list_as_row('log.csv', row_contents)

    print("logging event: reserved")
    if not args.quiet:
        print("process called with unexpected combo id %s and state 'Reserved'" % actor_name)
    exit(-1)


def main(args):
    """
    Python example actor

    This is an example how to use the `process` actor type to run a Python script.
    """

    if args.verbose is not None:
        if args.verbose == 1:
            print("verbose output enabled")
        elif args.verbose == 2:
            print("loud output enabled!")
elif args.verbose == 3:
            print("LOUD output enabled!!!")
        elif args.verbose > 4:
            print("Okay stop you're being ridiculous.")
            sys.exit(-2)
    else:
        args.verbose = 0
    # You could also check the actor name here and call different functions
    # depending on that variable instead of passing it to the state change
    # methods.

    new_state = args.state
    if new_state == "free":
        on_free(args, args.name)
    elif new_state == "inuse":
        on_use(args, args.name, args.userid)
    elif new_state == "tocheck":
        on_tocheck(args, args.name, args.userid)
    elif new_state == "blocked":
        on_blocked(args, args.name, args.userid)
    elif new_state == "disabled":
        on_disabled(args, args.name)
    elif new_state == "reserved":
        on_reserve(args, args.name, args.userid)
    else:
        print("Process actor called with unknown state %s" % new_state)

if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    # Parameters are passed to the Process actor as follows:
    # 1. the contents of params.args, split by whitespace as separate args
    # 2. the configured id of the actor (e.g. "DoorControl1")
    # 3. the new state as one of [free|inuse|tocheck|blocked|disabled|reserved]

    parser.add_argument("-q", "--quiet", help="be less verbose", action="store_true")
    parser.add_argument("-v", "--verbose", help="be more verbose", action="count")

    parser.add_argument("name",
                        help="name of this actor as configured in bffh.dhall"
                        )

    # We parse the new state using subparsers so that we only require a userid
    # in case it's a state that sets one.
    subparsers = parser.add_subparsers(required=True, dest="state")

    parser_free = subparsers.add_parser("free")
    parser_inuse = subparsers.add_parser("inuse")
    parser_inuse.add_argument("userid", help="The user that is now using the machine")

    parser_tocheck = subparsers.add_parser("tocheck")
    parser_tocheck.add_argument("userid", help="The user that should go check the machine")

    parser_blocked = subparsers.add_parser("blocked")
    parser_blocked.add_argument("userid", help="The user that marked the machine as blocked")

    parser_disabled = subparsers.add_parser("disabled")

    parser_reserved = subparsers.add_parser("reserved")
    parser_reserved.add_argument("userid", help="The user that reserved the machine")

    args = parser.parse_args()
    main(args)
```

```
chown -R bbfh:bffh /opt/fabinfra/adapters/csvlog/
```

Version #5
Erstellt: 7 Dezember 2024 00:59:03 von Mario Voigt (Stadtfabrikanten e.V.)
Zuletzt aktualisiert: 8 Dezember 2024 01:42:23 von Mario Voigt (Stadtfabrikanten e.V.)