

# Zdalne sterowanie zasilaczem HMP4040

---

## Spis treści:

1. [Zadanie](#)
  2. [Sterowanie zasilaczem przez USB](#)
  3. [Praca w sieci lokalnej](#)
    - [Serwer](#)
    - [Interfejs](#)
    - [Klient](#)
  4. [Praca z CANoe](#)
  5. [Formatowanie danych w projekcie](#)
  6. [Konfiguracja](#)
- 

## Zadanie

Stworzenie narzędzia umożliwiającego zdalne sterowanie zasilaczem HMP4040 w sieci lokalnej z wykorzystaniem graficznego interfejsu oraz przez CANoe.

## Sterowanie zasilaczem przez USB

Komunikacja z zasilaczem oparta jest na standardzie [SCPI](#) - dedykowanym rozwiązaniu do sterowania sprzętem laboratoryjnym z poziomu komputera.

Zestaw poleceń wykorzystywanych w dalszej części projektu:

Polecenie	Działanie
OUTP:GEN state	Zadanie stanu wyjścia zasilacza
INST:NSEL channel_id	Wybór kanału
OUTP:SEL state	Zadanie stanu wybranego kanału
APPL voltage, current	Zadanie parametrów wybranego kanału

Komunikację obsługuje biblioteka [pySerial](#). Podczas testowania połączenia występowały problemy z timeoutem zasilacza przy zbyt długim otwarciu portu szeregowego oraz zbyt częstym przesyłaniu poleceń. Aby wyeliminować problem przesyłanie wiadomości zostało zaimplementowane jako dekorator:

```
def _serial_handler(body):  
    def wrapper(self, *arg, **kw):  
        self.serial_id.open()  
        time.sleep(.015)  
        body(self, *arg, **kw)  
        time.sleep(.015)  
        self.serial_id.close()
```

Po otwarciu portu szeregowego występuje niewielkie opóźnienie gwarantujące, że port zostanie otwarty i przesłana zostanie poprawna wiadomość. Drugie opóźnienie zapobiega zamknięciu portu przed zakończeniem odbierania wiadomości przez zasilacz. Funkcja wywoływana jako `body()` generuje polecenie standardu SCPI.

## Praca w sieci lokalnej

Aby umożliwić dostęp do zasilacza z poziomu wielu urządzeń stworzona została hostowana w sieci lokalnej usługa. Składa się na nią serwer, interfejs webowy oraz klient, których działanie opisane jest w dalszej części.

### Serwer

Część serwerowa została stworzona przy pomocy frameworka [Flask](#). Jej zadaniem jest konwersja zapytań do poleceń standardu SCPI oraz renderowanie interfejsu webowego. Wbudowany we Flask serwer ze względu na słabą stabilność nie nadawał się do dalszej pracy. Jako alternatywa wykorzystany został serwer WSGI [Waitress](#).

### Interfejs

Interfejs graficzny dostępny jest z poziomu przeglądarki w sieci lokalnej. Pozwala na zadanie napięcia na dowolnym kanale oraz zmianę stanu dowolnego kanału i głównego wyjścia.

Poniżej grafika przedstawiająca wygląd webowego interfejsu stworzonego przy pomocy HTML oraz CSS:

The screenshot displays the 'ReMoTe4040' web interface. At the top, the title 'ReMoTe4040' is centered. Below it, there are five red rectangular buttons labeled 'CH1 0', 'CH2 0', 'CH3 0', 'CH4 0', and 'OUTPUT 0'. At the bottom, there are four control panels for channels CH1, CH2, CH3, and CH4. Each panel contains input fields for 'Voltage' and 'Current', and a 'set' button. The values shown are: CH1 (Voltage: 5.3, Current: 5.6), CH2 (Voltage: 4.8, Current: 1), CH3 (Voltage: 9.1, Current: 3.5), and CH4 (Voltage: 2.4, Current: 7.4).

Channel	Voltage	Current	Set Button
CH1	5.3	5.6	CH1 set
CH2	4.8	1	CH2 set
CH3	9.1	3.5	CH3 set
CH4	2.4	7.4	CH4 set

### Klient

Klient działa na zasadzie generowania zapytań wysyłanych na serwer przy pomocy biblioteki [Requests](#). Zapytania generowane są na podstawie zawartości pliku generowanego przez skrypt języka CAPL. Dodatkową funkcjonalnością klienta jest generowanie sztucznego ruchu - w określonych interwałach wysyłane są puste zapytania. Zapobiega to ewentualnemu przejściu serwera w tryb uśpienia.

### Praca z CANoe

CANoe za pomocą skryptu języka CAPL generuje zawartość pliku z informacją na temat numeru kanału, jego stanie oraz zadanych parametrach. Plik ten wczytywany jest przez klienta, który na jego podstawie generuje zapytania serwera.

Skrypt aktualizuje zawartość pliku przy rozpoczęciu ( on start ) i zakończeniu pomiaru ( on stopMeasurement ).

```
/*@!Encoding:1250*/
variables
{
    dword automationFile;
    char onCommand[81] = "{\"channel\": \"2\", \"state\": \"1\", \"params\": {\"voltage\": \"24.0\", \"current\": \"06.0\"}}";
    char offCommand[81] = "{\"channel\": \"2\", \"state\": \"0\", \"params\": {\"voltage\": \"24.0\", \"current\": \"06.0\"}}";
}

on start
{
    setFilePath("C:\\client", 1);
    automationFile = openFileWrite("capl_input.txt", 1);
    filePutString(onCommand, 81, automationFile);
    fileClose(automationFile);
}

on stopMeasurement
{
    setFilePath("C:\\client", 1);
    automationFile = openFileWrite("capl_input.txt", 1);
    filePutString(offCommand, 81, automationFile);
    fileClose(automationFile);
}
```

## Formatowanie danych w projekcie

Dla utrzymania prostoty w przetwarzaniu plików, w całym projekcie został zastosowany format typowy dla standardu [JSON](#). Pozwala on na wczytanie całego pliku jako słownika języka Python za pomocą poniższego kodu:

```
with open("file.txt", "r") as f:
    data = json.load(f)
```

Nadpisywanie pliku jest równie proste:

```
with open("file.txt", "w") as f:
    json.dump(data, f, indent=4, sort_keys=True)
```

## Konfiguracja

Stworzone zostały dwa pliki konfiguracyjne, które pozwalają na wprowadzanie zmian bez potrzeby edytowania samego kodu klient i serwera.

W przypadku serwera jest to:

```
|— automation_server
|   |— config.txt
```

Plik zawiera informacje o wyborze portu szeregowego oraz szybkości transmisji:

```
{"serial_port": "com4",  
"baudrate": "9600"}
```

Plik konfiguracyjny dla klienta znajduje się w:

```
|— client  
|   └─ config.txt
```

Zawiera jedynie możliwość zmiany adresu na jaki wysyłane są zapytania:

```
{"url": "http://127.0.0.1:8080"}
```

Aby skonfigurować skrypt CAPL należy zmodyfikować zmienne `onCommand` oraz `offCommand` z zachowaniem pierwotnego formatowania:

```
char onCommand[81] = "{\"channel\": \"2\", \"state\": \"1\", \"params\": {\"voltage\":  
\"00.0\", \"current\": \"00.0\"}}\"";  
char offCommand[81] = "{\"channel\": \"2\", \"state\": \"0\", \"params\":  
{\"voltage\": \"00.0\", \"current\": \"00.0\"}}\"";
```