Формат записи физиологических данных сессии для устройств Neiry.

Версия 3.0.

Версия 3.0 записи сессии использует формат хранения файлов **HDF5**.

В данной версии реализована запись сырых данных для каналов: ЭЭГ, ФПГ, МЭМС (акселерометр и гироскоп)

Содержимое файлов HDF5 организовано подобно иерархической файловой системе, и для доступа к данным применяются пути, сходные с POSIX-синтаксисом, например, */path/to/resource*. Метаданные хранятся в виде набора именованных атрибутов объектов.

Для индексации используются Б-деревья.

В версии 3.0 запись сессии происходит в один файл:

- **session_eeg.h5** - Сырые данные сигнала и метаинформация о датасетах, содержащих эти данные

session_eeg.h5

Фактические данные сеанса хранятся в потоке данных session.h5. Для обращения мы имеем две основные сущности:

Datasets

Название датасета, к которому мы можем обратиться, состоит из имени датчика, количества каналов и добавления в виде "_datasetName"

eeg4_datasetName

Таблица хранит значения временной метки и сигнала **ЭЭГ** по каждому каналу, каждое поле/строка/точка фиксируются каждые 0.004 секунды (частота дискредитации 250 Гц)

timestamp_name (миллисекунды с начала получения сигнала) (uint32)	**eeg1_name (float)**	**eeg2_name (float)**	**eeg3_name (float)**	**eeg4_name (float)**
…	…	…	…	…

accelerometer3_datasetName

Таблица хранит значения временной метки и сигнала **акселерометра** по каждому каналу, каждое поле/строка/точка фиксируются каждые 0.004 секунды (частота дискредитации **250 Гц**)

timestamp_name (миллисекунды с начала получения сигнала) (uint32)	**accelerometerX_name (float)**	**accelerometerY_name (float)**	**accelerometerZ_name (float)**
…	…	…	…

gyroscope3_datasetName

Таблица хранит значения временной метки и сигнала **гироскопа** по каждому каналу, каждое поле/строка/точка фиксируются каждые 0.004 секунды (частота дискредитации **250 Гц**)

timestamp_name (миллисекунды с начала получения сигнала) (uint32)	**gyroscopeX_name (float)**	**gyroscopeY_name (float)**	**gyroscopeZ_name (float)**
…	…	…	…

ppg1_datasetName

Таблица хранит значения временной метки и сигнала **фотоплетизмограммы** по каждому каналу, каждое поле/строка/точка фиксируются каждые 0.010 секунд (частота дискредитации **100 Гц**)

timestamp_name (миллисекунды с начала получения сигнала) (uint32)	**ppg1_name (float)**
…	…

Attribute

Артибут - это метаинформация по датасету, к которой мы можем обратиться по названию "%имя%_datasetAttribute", он содержит словарь с количеством каналов, их названиями и типом девайса. В случае датчика фпг, вместо названий каналов содержится информация о типе используемого датчика (red/ir) и его установленной интенсивности (0..100)

Также в атрибуте с названием “%имя%_datasetAttributeStartTime" лежит время старта сессии (нажатие кнопки старта сигнала и старта записи сигнала в h5 файл в формате unix timestamp в микросекундах)

Образец:

"eeg4_datasetAttributeStartTime" : 1678975312789087

"eeg4_datasetAttribute" : {'channelCount': 4, 'channelNames': ['O1', 'T3', 'T4', 'O2'], 'deviceType': 0}

Список девайсов:

NeiryHeadband = 0,
NeiryBuds = 1,
NeiryHeadphones = 2,
NeiryImpulse = 3,
Any = 4,
NeiryCallibri = 5,
SinWave = 100,
Noise = 101,
LSL = 102

Частоту дискретизации мы определяем по типу устройства, для Headband и Headphones у ЭЭГ, акселерометра и гироскопа она составляет 250 Гц, у ФПГ она составляет 100 Гц

Чтение с помощью Python

```
from pathlib import Path
import numpy as np
import h5py
import json

filename = r"./session_eeg.h5" # Path to file
filename = Path(filename)
datasets = []

with h5py.File(filename, "r") as f:
    for df_keys in f.keys():
        datasets.append(np.array(f[df_keys]))
        for attrs_keys in f[df_keys].attrs.keys():
            if type(f[df_keys].attrs[attrs_keys]) != np.int64:
                print(f"{df_keys} => {json.loads(f[df_keys].attrs[attrs_keys].decode('utf-8'))}")
            else:
                print(f"{df_keys} start time => {f[df_keys].attrs[attrs_keys]}", "\n")
print(datasets)
```

Пример вывода скрипта:

```
accelerometer3_datasetName => {'channelCount': 3, 'channelNames': ['X', 'Y', 'Z'], 'deviceType': 0}
accelerometer3_datasetName start time => 1694515290650564

eeg4_datasetName => {'channelCount': 4, 'channelNames': ['O1', 'T3', 'T4', 'O2'], 'deviceType': 0}
eeg4_datasetName start time => 1694515295150416

gyroscope3_datasetName => {'channelCount': 3, 'channelNames': ['X', 'Y', 'Z'], 'deviceType': 0}
gyroscope3_datasetName start time => 1694515290650564

ppg1_datasetName => {'channelCount': '1', 'detector': 'Red', 'intensity': '42', 'deviceType': 0}
ppg1_datasetName start time => 1694515290650530

[array([(    0, -0.3526719 , -0.9230018 , 0.12817775),
       (    4, -0.35193944, -0.92031616, 0.12988678),
       (    8, -0.35389262, -0.9230018 , 0.1264687 ), ...,
       (59540, -0.31580552, -0.94863737, 0.11059908),
       (59544, -0.3126316 , -0.9439985 , 0.10620441),
       (59548, -0.3170263 , -0.9427778 , 0.1081576 )],
      dtype=[('timestamp_name', '<u4'), ('accelerometerX_name', '<f4'), ('accelerometerY_name', '<f4'), ('accelerometerZ_name', '<f4')]), array([(    0, 0.08684951, 0.3774795 , 0.23152654, 0.124286  ),
       (    4, 0.08717867, 0.37468863, 0.23157136, 0.12431957),
       (    8, 0.08760816, 0.37254882, 0.2316933 , 0.12447778), ...,
       (65140, 0.10435315, 0.32009125, 0.22690009, 0.1320707 ),
       (65144, 0.10425512, 0.3197306 , 0.22682707, 0.13196728),
       (65148, 0.10422822, 0.31959534, 0.22681539, 0.13192265)],
      dtype=[('timestamp_name', '<u4'), ('eeg1_name', '<f4'), ('eeg2_name', '<f4'), ('eeg3_name', '<f4'), ('eeg4_name', '<f4')]), array([(    0,  0.       ,  0.       ,  0.       ,  0.       ),
       (    4,  0.       ,  0.       ,  0.       ,  0.       ),
       (    8,  0.       ,  0.       ,  0.       ,  0.       ), ...,
       (59540, -0.76296276, -0.65614796, -0.01525925),
       (59544, -0.76296276, -0.76296276, -0.06103702),
       (59548, -0.76296276, -0.6103702 , -0.10681479)],
      dtype=[('timestamp_name', '<u4'), ('gyroscopeX_name', '<f4'), ('gyroscopeY_name', '<f4'), ('gyroscopeZ_name', '<f4')]), array([(    0, 0.01492629), (   10, 0.01492725), (   20, 0.01492787), ...,
       (63650, 0.01570883), (63660, 0.01570931), (63670, 0.0157094 )],
      dtype=[('timestamp_name', '<u4'), ('ppg1_name', '<f4')])]
```

C++ представление

```
struct h5NeuroData4eegRelativeTimestamp {
    uint32_t timestamp;
    float eeg1;
    float eeg2;
    float eeg3;
    float eeg4;
};
struct h5NeuroData3accelerometerTimestamp {
    uint32_t timestamp;
    float accelerometerX;
    float accelerometerY;
    float accelerometerZ;
};
struct h5NeuroData3gyroscopeTimestamp {
    uint32_t timestamp;
    float gyroscopeX;
    float gyroscopeY;
    float gyroscopeZ;
};
struct h5NeuroData1ppgTimestamp {
    uint32_t timestamp;
    float ppg1;
};

....

 std::unique_ptr<H5::CompType> m_eeg4Type;

   m_eeg4Type = std::make_unique<H5::CompType>(sizeof(h5NeuroData4eegRelativeTimestamp));
   m_eeg4Type->insertMember("timestamp_name", HOFFSET(h5NeuroData4eegRelativeTimestamp, timestamp),
H5::PredType::NATIVE_UINT32);
   m_eeg4Type->insertMember("eeg1_name", HOFFSET(h5NeuroData4eegRelativeTimestamp, eeg1),
H5::PredType::NATIVE_FLOAT);
   m_eeg4Type->insertMember("eeg2_name", HOFFSET(h5NeuroData4eegRelativeTimestamp, eeg2),
H5::PredType::NATIVE_FLOAT);
   m_eeg4Type->insertMember("eeg3_name", HOFFSET(h5NeuroData4eegRelativeTimestamp, eeg3),
H5::PredType::NATIVE_FLOAT);
   m_eeg4Type->insertMember("eeg4_name", HOFFSET(h5NeuroData4eegRelativeTimestamp, eeg4),
H5::PredType::NATIVE_FLOAT);

   m_accelerometer3Type->insertMember("timestamp_name", HOFFSET(h5NeuroData3accelerometerTimestamp,
timestamp), H5::PredType::NATIVE_UINT32);
```

m_accelerometer3Type->insertMember("accelerometerX_name", HOFFSET(h5NeuroData3accelerometerTimestamp, accelerometerX), H5::PredType::NATIVE_FLOAT);
m_accelerometer3Type->insertMember("accelerometerY_name", HOFFSET(h5NeuroData3accelerometerTimestamp, accelerometerY), H5::PredType::NATIVE_FLOAT);
m_accelerometer3Type->insertMember("accelerometerZ_name", HOFFSET(h5NeuroData3accelerometerTimestamp, accelerometerZ), H5::PredType::NATIVE_FLOAT);

m_gyroscope3Type->insertMember("timestamp_name", HOFFSET(h5NeuroData3gyroscopeTimestamp, timestamp), H5::PredType::NATIVE_UINT32);
m_gyroscope3Type->insertMember("gyroscopeX_name", HOFFSET(h5NeuroData3gyroscopeTimestamp, gyroscopeX), H5::PredType::NATIVE_FLOAT);
m_gyroscope3Type->insertMember("gyroscopeY_name", HOFFSET(h5NeuroData3gyroscopeTimestamp, gyroscopeY), H5::PredType::NATIVE_FLOAT);
m_gyroscope3Type->insertMember("gyroscopeZ_name", HOFFSET(h5NeuroData3gyroscopeTimestamp, gyroscopeZ), H5::PredType::NATIVE_FLOAT);

m_ppg1Type->insertMember("timestamp_name", HOFFSET(h5NeuroData1ppgTimestamp, timestamp), H5::PredType::NATIVE_UINT32);
m_ppg1Type->insertMember("ppg1_name", HOFFSET(h5NeuroData1ppgTimestamp, ppg1), H5::PredType::NATIVE_FLOAT);

Подсчет данных NeuroFeedBack (NFB) и их запись

Получение метрик

После получения сырых данных с частотой дискретизации 250 Гц и их последующей обработки мы получаем наши NFB-метрики с частотой 10 Гц (раз в 0,1 секунду == 100 миллисекунд)

Запись в XLSX

Запись в XLSX файл происходит по строкам, каждую из которых мы получаем в результате усреднения данных за последние 30 секунд, то есть частота получения строк это 1/30 Гц (раз в 30 секунд)

Разметка

Описание

Разметка, которую пользователь проводит вручную, сохраняется в файл **json** в определенном формате:

Изначально у нас есть пустой массив [], который заполняется словарями с активностями и интервалами времени, когда они проходили

Возможные активности:

- *1*
- *2*
- *3*
- *4*
- *5*

- *calibrationiapf*
- *calibrationbaseline*

В каждом словаре содержится два ключа:

1. **‘activity'**, по которому лежит наименование активности, если та присутствовала во время сессии
   - Если за время сессии какой-то активности из возможных не происходило, то соответствующий словарь не создается

2. **‘times'**, по которому лежит массив словарей с ключами **’start_time'** и **‘end_time’**.
   - Каждый такой словарь соответствует какому-то временному промежутку, во время которого происходила активность
   - По ключам **’start_time'** и **‘end_time’** лежат, соответственно, время начала активности и время её конца в формате unix timestamp в микросекундах