# MNE-Python: Enhancing Performance of Signal Browsing using PyQt

Google Summer of Code 2021 - Martin Schulz

Mentors: Clemens Brunner, Daniel McCloy

## Table of Contents

## About me

Since 2015 I am studying medicine (graduation planned for 2023) at the University of Heidelberg, Germany (UTC+2) where I also started my medical thesis in 2018 in an MEG-Lab with PD André Rupp about pain processing. This is where I first got in contact with MNE-Python and programming in general. To facilitate my work and the work of others in the laboratory with MNE-Python, I developed a PyQt-based GUI for MNE-Python. In 2020 I first contributed to MNE-Python (alias marsipu). Recently I participated in the 2021 MNE Code Sprint which motivated me for this application.
If you want to learn more about me, please read my CV.

## Code Contribution

- Exemplary PR to MNE-Python
- Full list of PRs to MNE-Python

# Project

## Sub-org name

MNE-Python (Website, GitHub)

## Project Abstract

This project's objective is to provide an additional backend for the **visualization of two-dimensional data** (channels x time) using **PyQt**. It is supposed to offer **high performance** for visualizing data and thus facilitating signal inspection.

First of all the most suitable PyQt-based backend will be determined regarding performance and compatibility under the specific conditions of MNE-Python. The chosen backend will then be used to create an implementation with equivalent features to the original matplotlib-version. Finally the foundation for an integration of the new implementation alongside the original version will be laid.

## Detailed Description

Inspecting two-dimensional data (channels x time, also called “Raw” in MNE-Python terminology) is an integral part of the analysis of electrophysiological data. During this step artifacts can be annotated, bad channels marked, and even preliminary analysis of the data can be done (by inspecting the time course or analyzing frequency components). In MNE-Python, the visualization is currently based on matplotlib with its various backends. While it offers a robust and proven solution, it has some limitations regarding performance and visualization. For example, it does not offer smooth scrolling through the data and when plotting a larger number of channels and time points simultaneously, performance decreases. Furthermore, the GUI elements and slots for interactivity (e.g. buttons) are limited and do not follow platform look-and-feel. A PyQt-based solution should offer more flexibility for user-interaction.

The project's objective is to provide an alternative to the existing matplotlib-based signal browser. As suggested by Clemens Brunner, a solution based on PyQt[1] could offer the desired increase in performance. The package pyqtgraph demonstrates the capabilities of a plotting backend based on a pure PyQt solution. Using it as a dependency for the Raw Browser could provide an already polished GUI framework and therefore easier implementation, but it could also be difficult to adapt the functionality to the specific needs of MNE-Python. A custom plotting-backend could offer a more tailored solution but it could prove to be a greater challenge. With SigViewer there already exists a C++-Implementation of a custom plotting backend, which could serve as a blueprint for a PyQt-version in a Raw Browser without additional dependencies.

[1] PyQt stands here for Python bindings for Qt, we aim for compatibility with PyQt5/6 and PySide2/6 probably using qtpy.

The first part of the project would be to further develop existing prototypes (custom Qt-backend, pyqtgraph, by Clemens Brunner) of a Raw Browser and to compare them regarding:

- Scrolling Speed
- Scaling (dynamic resampling for improved performance)
- Dynamic change of layout (number of samples/channels)
- Layout (vertical markers, customizable scrollbar)
- (Area-)Selection (for annotations)

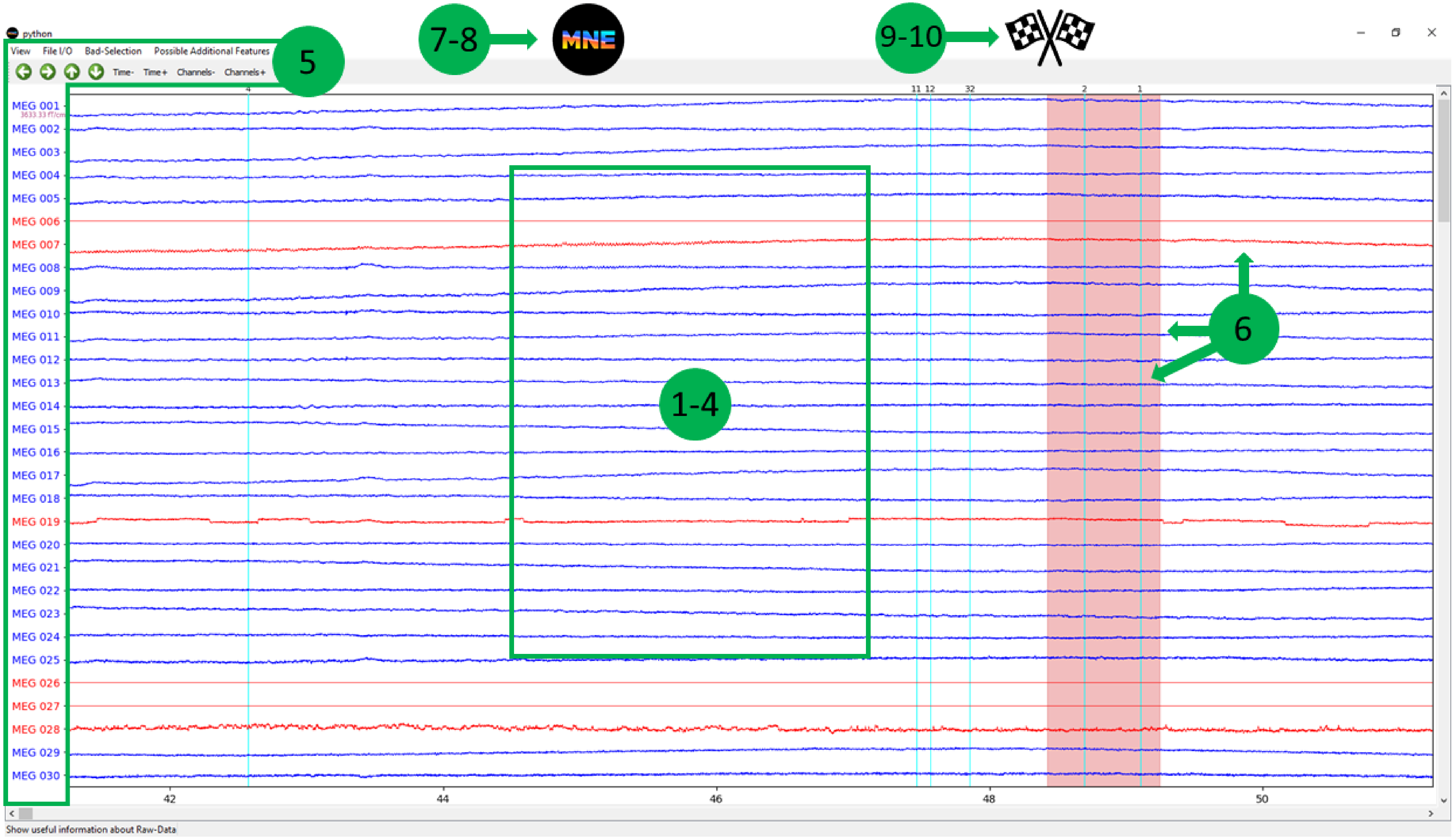
The next step will be choosing the best solution to build the future RawBrowser-Backend. The decision will be based on performance-tests and overall practicability and compatibility to MNE-Python-requirements.

Then, the interface will be added for controlling the plot. The visualization will be adjusted to properly represent electrophysiological data. User experience will be modeled close to the original matplotlib-implementation by reproducing its features for interaction with the data such as marking bad-channels, placing annotations, adding a vertical marker etc..

Finally, the new backend will be wrapped into a class to provide the required programming interface for an integration into MNE-Python as an alternative for Raw.plot(). It will be ensured by adapting/creating tests for the new backend that user interaction leads to expected results and performance is maintained under multiple possible circumstances (e.g., different file formats, file sizes). A custom image-retriever (scraper) for sphinx-gallery would allow the new backend to also appear inside the tutorials/examples on the MNE-Python-Website.

Overall, the minimal outcome should be a fundamental version of a Raw Browser with PyQt, which showcases its advantages over the original implementation. At best, it provides equivalent features and can be implemented into the main-branch.

In case of spare time the new backend could be extended to the plots of Epochs and ICA-Sources, which in MNE-Python have similar plotting interfaces and similar interactivity to the Raw Browser.



*A schematic overview of the steps of the project (based on the original matplotlib RawBrowser, **numbers=weeks**)*

# Timeline

## Time commitment

I will spend 4-6h/day on this project. Additionally I intend to spend some extra time each day (~1-2h/day) participating in the MNE-Python community in their forum, and/or by addressing small, unrelated issues that arise.

## Community Bonding (17.05. – 07.06.2021)

I already spent a week contributing at the MNE-Code-Sprint (15-19 March 2021), where I got to know the core developers. So I will use the community bonding period to follow more closely the ongoing work related to visualization and communicate with the developers who are most knowledgeable about it. Besides I will learn more about the usage of the different candidates envisioned for the backend.

### Learn about

- Custom Qt Plots
- SigViewer-Implementation
- Using PyQtGraph
- (Existing) Matplotlib-based Raw Browser

### Discussion

- Open an issue on GitHub for discussion about the new RawBrowser-backend
- Exchange with other developers about visualization

## Coding (07.06. - 16.08)

### Week 1 (07.06. - 11.06.): Create Prototypes 1

- Qt:
  - Create Line-Object
  - Create Multi-Line-Layout
  - Time/Channel-Scrolling

**Deliverable**: custom Qt-prototype

### Week 2 (14.06. - 18.06.): Create Prototypes 2

- PyQtGraph: Create prototype comparable to the Qt-prototype from week 1
- Add capabilities for both:
  - Changing number of samples/channels
  - Performance-Improvements (Visual Resampling)

**Deliverable**: Improved prototypes

## Week 3 (21.06. - 25.06.): Create Prototypes 3

- Amplitude-Scaling
- Simple Tests for later capabilities:
    - area-selection (for annotations)
    - vertical line (vertical guide, events)
    - customizable scrollbars

**Deliverable**: Further improved prototypes

## Week 4 (28.06. - 02.07.): Compare Prototypes

- Finish tasks left from previous weeks which are required for comparison
- Create Tests for Performance-Comparison
- Compare Prototypes and choose the one most suitable

**Deliverable**: Backend-Decision

## Week 5 (05.07. - 09.07.): Interface

- Axes: Channels x Time
- Menu and Toolbar
- File I/O
- Keybindings

**Deliverable**: RawBrowser with Interface-Elements

## Week 6 (12.07. - 16.07.): Interactivity

- Mark bad channels
- Annotate bad segments
- Vertical Guide/Events
- optional:
    - Butterfly-Mode
    - DC removal

**Deliverable**: interactive features (as in original matplotlib-backend) added

### Week 7 (19.07. - 23.07.): Implementation

- Create Class which fits into structure of Raw.plot()
- Implementation into MNE-Python
- Add/Edit Tests

**Deliverable**: Usage possible from MNE-Python (on my experimental branch)

### Week 8 (26.07. - 30.07.): Tests & Documentation

- Continue Add/Edit Tests
- Add compatibility for existing tutorials with sphinx-gallery

**Deliverable**: Tests & Documentation

### Week 9 (02.08. - 06.08.): Buffer for unfinished tasks

**Deliverable**: Finished unfinished tasks

### Week 10 (09.08. - 13.08.): Submitting the Project

- Open Pull-Request on GitHub
- Resolve upcoming issues

**Deliverable**: Finished Project

## Other commitments

On two weekends during the program (05.06. - 06.06, 10.07. - 11.07.) I will participate in courses for emergency medicine which require preparation time in the week before. However this should not interfere with my ability to participate in GSoC those weeks.