

Hoksotin Project

Kari Aliranta
Jaakko Leppäkangas
Janne Pesonen
Atte Rautio

Application Report

Public
Version 1.0.0
2013-06-28

University of Jyväskylä
Department of Mathematical Information Technology
Jyväskylä
Finland

Acceptor	Date	Signature	Clarified signature
Manager	___.2013		
Customer	___.2013		
Supervisor	___.2013		

Document Info

Authors

- Kari Aliranta (KA) kari.p.aliranta@student.jyu.fi +358405736683
- Janne Pesonen (JP) janne.o.pesonen@student.jyu.fi +358400894370

Document name: Hoksotin Project, Application Report

Page count: 40

Abstract: Hoksotin project developed a prototype graphical frontend for integrating the applications and command line scripts used in MEG data analysis. The document describes the backgrounds of the software as well as presents the user interface and the application structure. Programming and testing practices are also presented. The realization of the functional requirements and other objectives as well as advice for future developers are also covered.

Keywords: Application structure, future development, magnetoencephalography, MEG, MNE, programming practices, Python, PyQt, realization of objectives, testing, user interface.

History of Changes

Version	Date	Modifications	Committers
0.0.1	2013-05-13	Writing the document started.	KA
0.0.2	2013-05-15	Introduction chapter was written.	KA
0.0.3	2013-05-16	Terminology chapter was written.	KA
0.0.4	2013-05-24	Most of the text in UI chapter was written.	KA
0.1.0	2013-05-27	The UI chapter was clarified.	KA
0.1.1	2013-05-28	More figures were added to the the UI chapter.	KA
0.1.2	2013-05-30	Several styles were tweaked.	KA
0.1.3	2013-06-10	UI chapter was preliminarily finished.	KA
0.1.4	2013-06-10	General Application Structure chapter was started.	KA
0.1.5	2013-06-12	Chapters about Realization of Objectives, and Guide for Future Developers were added.	JP
0.2.0	2013-06-12	ToC and headings were updated.	KA
0.2.1	2013-06-17	UI chapter was updated and several figures were added.	KA
0.2.2	2013-06-18	General Application Structure chapter was added.	KA
0.3.0	2013-06-18	Testing was moved to Programming and Testing Practices chapter.	KA
0.3.1	2013-06-18	Most of the Realization of Objectives chapter was rewritten.	KA
0.3.2	2013-06-21	References chapter was added and Guide for Future Developers chapter was restructured.	KA
0.4.0	2013-06-24	Chapter about background and a summary was written.	KA
0.5.0	2013-06-25	New figures were added in the UI section.	KA
1.0.0	2013-06-28	Some typos were corrected.	KA

Project Information

Hoksotin project created a prototype user interface for preprocessing, analyzing and visualizing MEG data.

Project members:

Aliranta Kari	kari.p.aliranta@student.jyu.fi	+358405736683
Leppäkangas Jaakko	jaeilepp@student.jyu.fi	+358445508574
Pesonen Janne	janne.o.pesonen@student.jyu.fi	+358400894370
Rautio Atte	atte.m.rautio@student.jyu.fi	+358408409084

Representative of the customer:

Parviainen Tiina	tiina.m.parviainen@jyu.fi	+358408053533
------------------	---------------------------	---------------

Supervisors:

Puoliväli Tuomas	tuomas.a.b.puolivali@student.jyu.fi	
Santanen Jukka Pekka	jukka-pekka.santanen@mit.jyu.fi	+358408053299

General contact information for the project:

Mailing lists	hoksotin@korppi.jyu.fi, hoksotin_opetus@korppi.jyu.fi
Mailing list archives	https://korppi.jyu.fi/kotka/servlet/list-archive/hoksotin/ https://korppi.jyu.fi/kotka/servlet/list-archive/hoksotin_opetus/

Contents

1 Introduction.....	1
2 Terminology Related to the Domain and the Application.....	2
2.1 Domain Terminology.....	2
2.2 Terminology Related to Software Development in General.....	3
3 Background and Goals of Hoksotin Project.....	5
4 The User Interface of the Application.....	6
4.1 Choosing a Workspace and Creating an Experiment.....	6
4.2 Loading an Existing Experiment.....	9
4.3 The Main Window of the Application.....	9
4.4 The Menu Bar.....	10
4.5 Raw Tab.....	10
4.6 Preprocessing Tab.....	10
4.6.1 MaxFilter.....	11
4.6.2 Calculate ECG Projections.....	11
4.6.3 Calculate EOG Projections.....	14
4.6.4 Apply ECG Projections and Apply EOG Projections.....	14
4.6.5 Magnitude Spectrum.....	14
4.7 Epoching Tab.....	14
4.7.1 Event Selection Dialog.....	15
4.7.2 Epoch Creation Dialog.....	16
4.8 Averaging Tab.....	18
4.9 TFR Tab.....	19
4.9.1 TFR Visualization.....	21
4.9.2 TFR Topology Visualization.....	23
5 General Application Structure.....	25
5.1 Formats, Components and Software.....	25
5.2 Class Structure.....	26
5.3 File and Data Formats.....	27
6 Programming and Testing Practices.....	29
6.1 Formatting, Naming and Commenting Practices.....	29
6.2 Source Code Example.....	30
6.3 Grouping Practices.....	31
6.4 Development Platform.....	31
6.5 Testing Practices and Results.....	31
7 Realization of Objectives.....	33
7.1 Realization of Requirements.....	33
7.2 Unsatisfactory Solutions in the Implementation.....	33
7.3 Challenges in the Implementation.....	34
7.4 Modifications During the Implementation.....	35

8 Guide for Future Developers.....	36
8.1 Essential Bugs.....	36
8.2 Improvements to Existing Features.....	36
8.3 Further Development Ideas.....	37
8.4 Improvements to Development Practices.....	37
9 Summary.....	38
10 References.....	39

1 Introduction

MEG (magnetoencephalography) is a method of measuring magnetic fields caused by electrical activity in the brain. By analyzing the data accumulated by the measurements, the timing and location of the brain activity can be discerned. In a common type of MEG experiment setting the test subject is given sensory input, i.e. shown pictures or played sounds to, and the activity of the subject's brain is measured by the equipment.

The data accumulated by the measuring instruments needs to be preprocessed and analyzed in order to yield useful information. Presently, there exist pieces of software capable of doing this preprocessing and analyzing, but they don't always integrate well with each other nor are always very usable. Specifically, they don't make the process very clear for a still learning, non-expert user, and are occasionally burdensome to use for an advanced user, too.

Hoksotin project developed a prototype graphical frontend, called Meggie, for integrating some applications and command line scripts used in MEG data analysis. The main backend for the frontend is the open source MNE software package [1], which is hosted at the Martinos Center for Biomedical Imaging at Harvard-MIT. The prototype has an option to handle part of the preprocessing using MaxFilter software by Elekta Neuromag, a manufacturer of MEG equipment. The frontend is programmed with Python 2.7 and uses PyQt4 as the graphical user interface library.

The application report mainly describes the solutions for the requirements described in the Software Requirements Specification [2]. The user interface, general structure, testing, known defects and possible future development ideas are also covered. The information related to the project itself can be found in the Project Report [3], whereas the detailed Class Documentation [4] includes more details about the class structure of the application. The Installation Guide [5] is also a separate document, covering the installation of the dependencies of the application.

Chapter 2 describes the terminology related to the domain and the developed application. Chapter 3 elaborates on the background of the application and the process it is designed to facilitate. Chapter 4 is dedicated to the user interface of the application, whereas Chapter 5 describes the general structure of the application. The programming and testing guidelines of the project are briefly outlined in Chapter 6, and Chapter 7 is a reflective chapter about the attained and unattained goals of the project. Chapter 8 suggests possible future paths of development.

2 Terminology Related to the Domain and the Application

The chapter explains the essential terms that appear in the document.

2.1 Domain Terminology

The terms below are specific to Hoksotin project and describe some essential concepts related to its domain.

Artefact	is a disturbance in the measured MEG signal caused by electrical activity originating from outside the brain.
ECG channel	i.e. electrocardiogram channel, is a channel in the measured MEG data, dedicated to measuring the electrical activity of the heart either by a dedicated electrode or a chosen MEG channel. It is used to filter the data in order to remove the disturbances caused by the said electrical activity.
EEG	i.e. electroencephalography, is a technique for examining the electrical activity of the brain directly via electrodes measuring the electrical currents reaching the surface of the skull.
EOG channel	i.e. electrooculogram channel, is a channel in the measured MEG data, dedicated to measuring the electrical activity caused by blinking the eyes. It is used to filter the data in order to remove the disturbances caused by the said electrical activity.
Epoch	is a timeslice of MEG data. In data containing triggers, out of which events can be created, it is generally defined to begin a few hundred milliseconds before an event and to end a few hundred milliseconds after the said event.
Event	is a point in time in the signal data, corresponding to a single trigger point on the trigger channel. Not all triggers are chosen as basis for events in a given experiment. Epochs are derived from events.
MC	i.e. Motion Correction, is an MEG data preprocessing technique used for countering the disturbances in the signal caused by the movements of subject's head during the measurement.
MEG	i.e. magnetoencephalography, is a technique for examining the electrical activity of the brain by measuring the magnetic

	fields caused by the said electrical activity.
MEG channel	is a portion of MEG signal, measured by a single SQUID sensor.
Preprocessing	is the first step in analyzing the MEG data, consisting of filtering out the unwanted disturbances. It generally includes SSS and possibly tSSS and/or MC. It may also include removal of artifacts based on the data in the ECG and EOG channels.
Raw file	is the first file generated from the information measured by the MEG measuring equipment. It includes unprocessed signal and trigger information. Meggie application understands only binary raw file format of type fiff.
SQUID	i.e. Superconducting Quantum Interference Device, is a type of sensor used in the MEG system. Contemporary MEG systems include hundreds of individual SQUIDS.
SSS	i.e. Signal Space Separation, is a signal data preprocessing technique used to distinguish: 1) the signal data originating from the brain and 2) the noise from outside the MEG equipment and 3) the noise caused by MEG equipment itself, e.g. the SQUIDS.
Stimulus	is a specifically defined outside input to the test subject, the brain response to which is monitored with the MEG equipment. It is often a sound or a picture. It is often used interchangeably with the word trigger, even though trigger is a technical term related to the trigger channel, and may or may not correspond to a stimulus.
Trigger channel	is an MEG channel with temporal information about triggers. Triggers may correspond to stimuli, or they may be other time points of importance in the experiment (e.g. test subject clicking a button).
TSSS	i.e. Temporal Signal Space Separation, is an extension of SSS. It is used for removal of disturbances originating from inside the test subject's body, e.g. pacemakers or braces.

2.2 Terminology Related to Software Development in General

The following terms are related to Hoksotin project, but are generally used in software development and could apply to many other projects as well. See Chapter 5.1 starting from page 25 for specific technologies related to Meggie.

Doxygen	is a tool for generating class documentation for applications written in various programming languages, including Python.
Eclipse	is an open source integrated development environment (IDE). An Eclipse plugin for Python development, know as PyDev, is also available and was used for the development of Meggie.
PyQt	is a user interface library that includes Python bindings for the Qt graphical interface toolkit. It is available as free software and is hosted by Riverbank Computing.
Python	is an open source programming language, suitable for object-oriented programming paradigm, among others.
YouSource	is a web-based source code collaboration system developed in the University of Jyväskylä. It uses Git as the version control system. Hoksotin project used YouSource as its development repository.

3 Background and Goals of Hoksotin Project

Hoksotin project was started off from the need to create an user interface for preprocessing, analyzing and visualizing magnetoencephalography (MEG) data. The chapter briefly introduces the background and initial needs that led to starting of Hoksotin project.

As stated in Chapter 1, MEG is a method of examining brain activity by measuring the magnetic fields generated by the electrical activity in the brain. The measuring instruments generate copious amounts of raw data, which requires several steps of preprocessing, refinement and analysis in order to yield useful information.

Currently, there are open source alternatives for graphical user interfaces in MEG data analysis, but none of these alternatives concentrate primarily on MEG. Closed source alternatives also exist, but they do not cover every step of the process and require tedious manual managing of the files generated in the steps. Existing open source scripts and libraries for MEG analysis, specifically MNE libraries, are capable of very detailed data processing and analysis, but mostly lack any graphical user interface whatsoever.

At the moment, the researchers in the MEG field utilize a mix of closed and open source software, graphical user interfaces and command line scripts. The process typically starts with the preprocessing step, which the researcher starts by employing basic noise removal techniques like SSS and TSSS via a closed source graphical user interface application. She/he then proceeds to remove disturbances caused by heartbeat and blinks by calling open source libraries via command line scripts. Having removed the unwanted noise from the data, she/he may then continue on the way of the command line, analyzing the data with several commands to open source libraries. The commands require long lists of parameters and are not saved for future use except via copy pasting and command history of the terminal program used. The process is obviously not very intuitive and presents a steep learning curve for an uninitiated person, such as a student learning the field.

The main goal of the Hoksotin project was, therefore, to develop a graphical user interface prototype to integrate the functionality of the open source libraries and scripts. Where applicable, as in the case of MaxFilter, optional integration of closed source software was also taken into account. The prototype interface was designed to guide an inexperienced user, clearly denoting the most common sequences of methods and paths of analysis. At the same time it was designed to be flexible enough to allow the experienced user to perform advanced analysis without resorting to manual managing of parameters and files.

4 The User Interface of the Application

The user interface of Meggie application is based on a single persistent main window. Presently, the main window only represents the preprocessing and analysis process of a single raw file at a time. The representation is called an **experiment**, and the various stages of the experiment are divided into tabs in the user interface.

Initially, the main window is empty, but after creating a new experiment (see Section 4.1) or loading an existing experiment (see Section 4.2) the tabs representing the stages appear, allowing various methods of preprocessing and visualization (see Figure 4).

On the technical side, the application uses PyQt4 as the user interface library, allowing the usage of most Qt widgets. This in turn allows the application to integrate with the (Qt) theme of the desktop environment in use.

PLEASE NOTE: the chapter assumes the reader is familiar with the basics of MEG data analysis. Additionally, many of the dialogs of the application pass the values of their parameter fields to MNE-Python. In these cases, the descriptions of these parameter fields closely follow the MNE-Python reference guide [6].

4.1 Choosing a Workspace and Creating an Experiment

When the application is launched, the user is presented with an empty window with toolbar commands allowing the creation of a new experiment or loading a pre-existing one (see Figure 1).

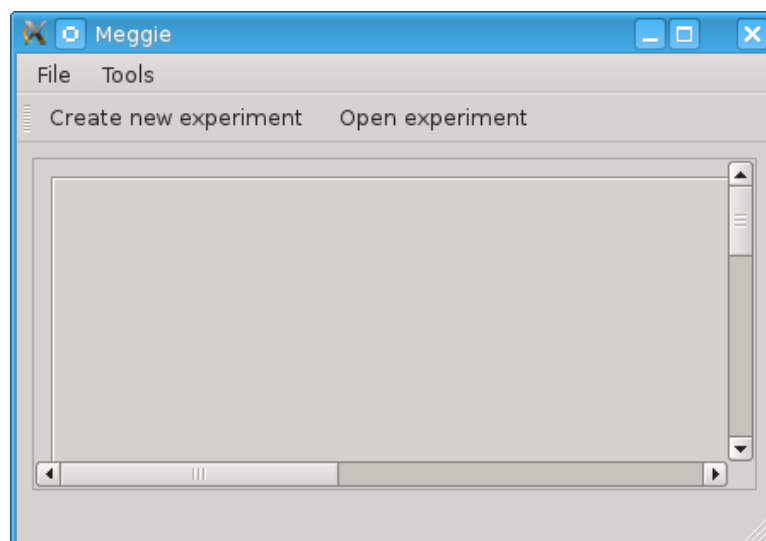


Figure 1: Main window without an experiment loaded.

Should the user choose the command *Create new experiment*, the application prompts the user to choose a workspace directory for the application (see Figure 2). The workspace directory is the directory where all the experiment files created by the application will be saved. The directory will be prompted only once, subsequently the application will use the chosen directory. The workspace directory can be later changed by clicking the *Set workspace* command in *File* menu (see Section 4.4).

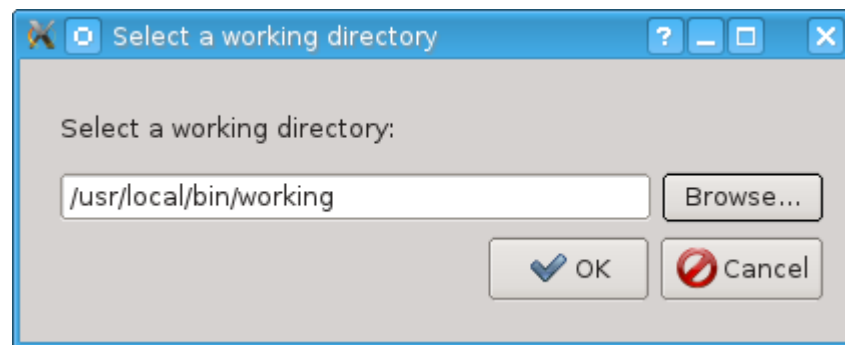


Figure 2: Workspace chooser dialog.

After the workspace has been chosen, the user is required to click *Create new experiment* button (see Figure 1) again to actually create a new experiment. The user is then presented the experiment creation dialog (see Figure 3).

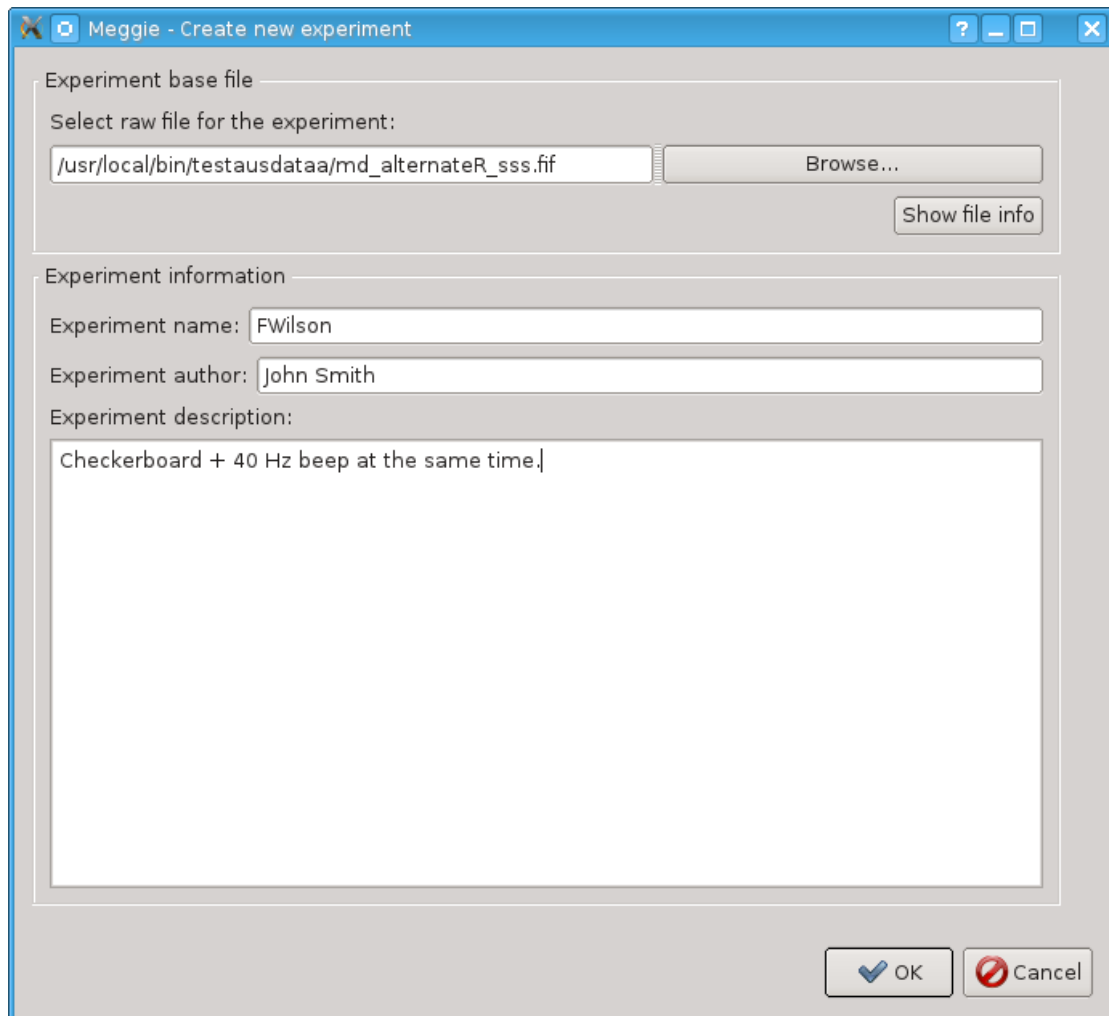


Figure 3: Experiment creation dialog.

Browse button opens a standard file chooser dialog which allows the user to navigate to the desired file. The text box in front of the button allows the direct input of the absolute path to the desired file. The other text boxes and the text are specify metadata for the experiment. After clicking *OK* button, the user is taken to the main window (see Figure 4) populated with the information about the raw file loaded.

PLEASE NOTE: Meggie creates a directory for the experiment, named after *Experiment name* specified in the corresponding text box. The directory contains all the files related to the experiment, and resides in the workspace directory of the application.

4.2 Loading an Existing Experiment

An existing experiment is loaded by clicking the *Open experiment* toolbar button in the main window (see Figure 1). The button opens a standard file browser, allowing the user to navigate to the directory of the existing experiment. The experiment is then loaded by clicking the *Open* button of the file browser.

4.3 The Main Window of the Application

The main window of the application (see Figure 4) includes the **menu bar**, the **tool bar**, the **tab area** and the **status bar**. The tab area represents the different steps of MEG data analysis as tabs. The status bar shows the path to the current working file.

The tabs are generally divided into two parts. The upper part includes either uneditable information (see Raw tab in Figure 1), information about the completed steps of the preprocessing (see Preprocessing tab in Figure 5) or a list of epoch collections (see Averaging tab in Figure 12 and TFR tab in Figure 14). The lower part always includes the available actions and methods for each step.

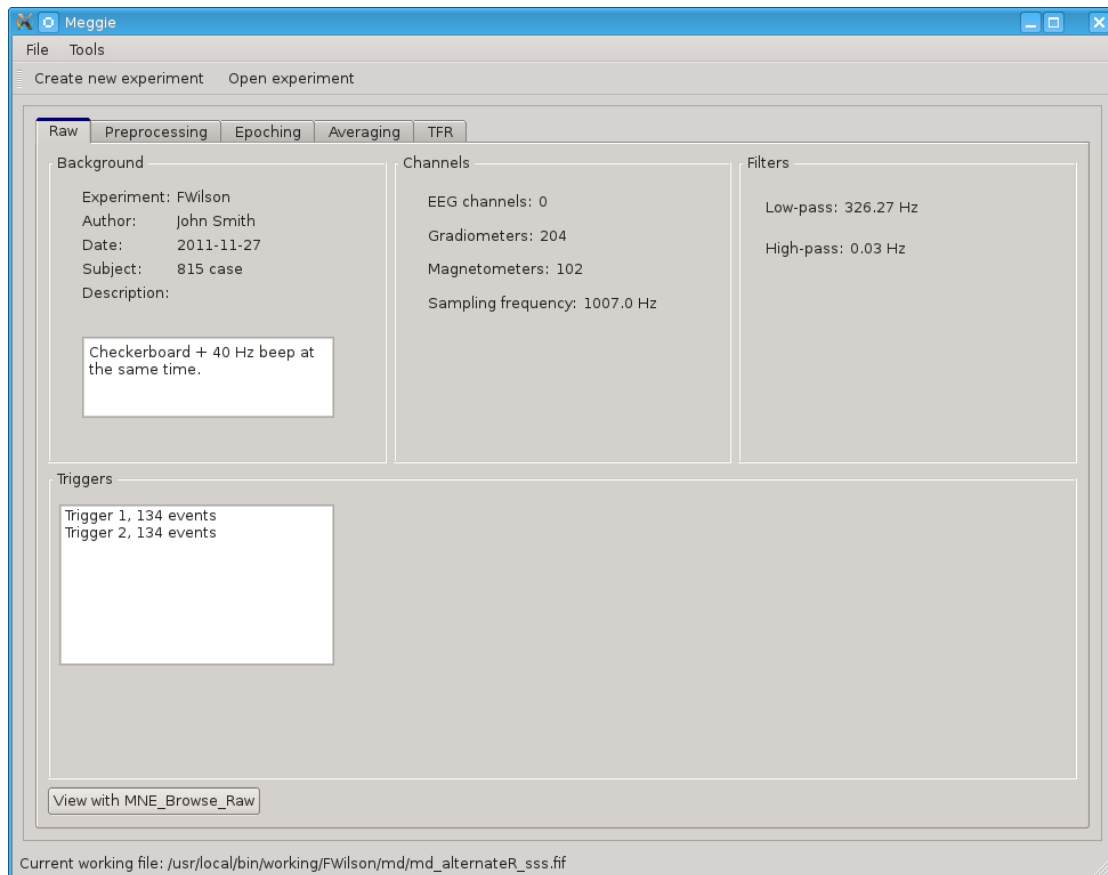


Figure 4: Main window with the raw tab active.

4.4 The Menu Bar

File menu includes the commands to *Create new experiment* and *Open experiment*. These work exactly the same way as their toolbar siblings described in Section 4.1. Additionally, the menu includes the command *Set workspace*. It opens the workspace chooser dialog illustrated in Figure 2. The file menu also has a command to show *About* dialog, including the license information about the application.

Tools menu has a single command, *Preferences*. Preferences dialog currently includes a simple list of equipment calibration files provided by the MaxFilter program, ordered by the location of the equipment. The user can then choose the desired calibration file for the the experiment. If MaxFilter is not available, the dialog simply shows an empty list.

4.5 Raw Tab

The raw tab (see Figure 4) is the default tab to be shown after the creation or opening of an experiment. It shows basic information about the raw file of the experiment. This information includes experiment information set by the user at the time of creation of the experiment and information read from the raw file info fields themselves (with `mne.raw.info` command). The raw file can also be viewed with an external browser by clicking the button *View with MNE_browse_raw*.

4.6 Preprocessing Tab

The preprocessing tab (see Figure 5) includes options to apply *MaxFilter* methods SSS, tSSS and MC to the raw file. It also allows calculating SSP/PCA projections for ECG and EOG artifacts and viewing *Magnitude spectrum*. The completed processing steps are indicated by the green checkmarks in the box *Preprocessing steps completed*.

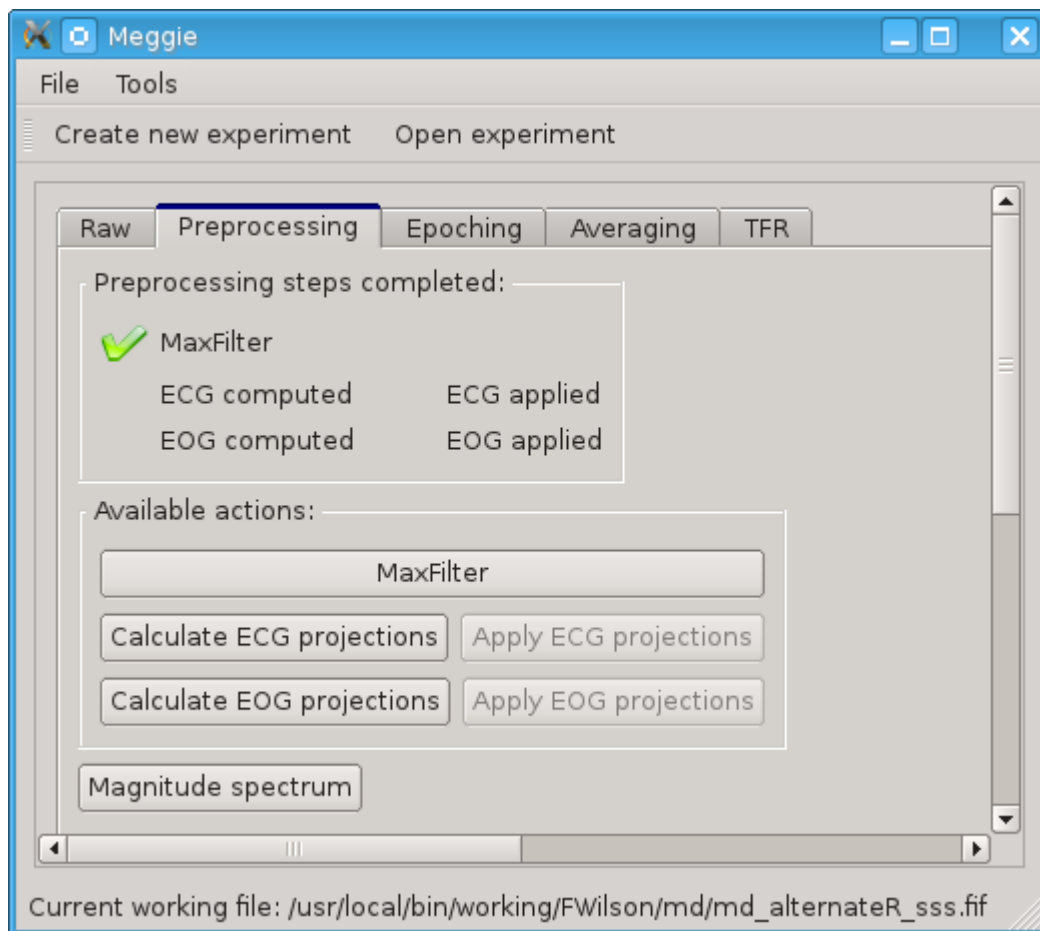


Figure 5: Main window with the preprocessing tab active.

4.6.1 *MaxFilter*

The *MaxFilter* button opens a dialog which includes parameters to call the proprietary *MaxFilter* program with. The fields of the dialog closely follow the nomenclature of *MaxFilter* parameters. For parameters and their descriptions, see the *MaxFilter* manual [7].

4.6.2 *Calculate ECG Projections*

The *Calculate ECG Projections* button opens the dialog (see Figures 6 and 7) for calculating ECG projections for the raw file. The MNE method used is `mne.preprocessing.compute_proj_ecg`.

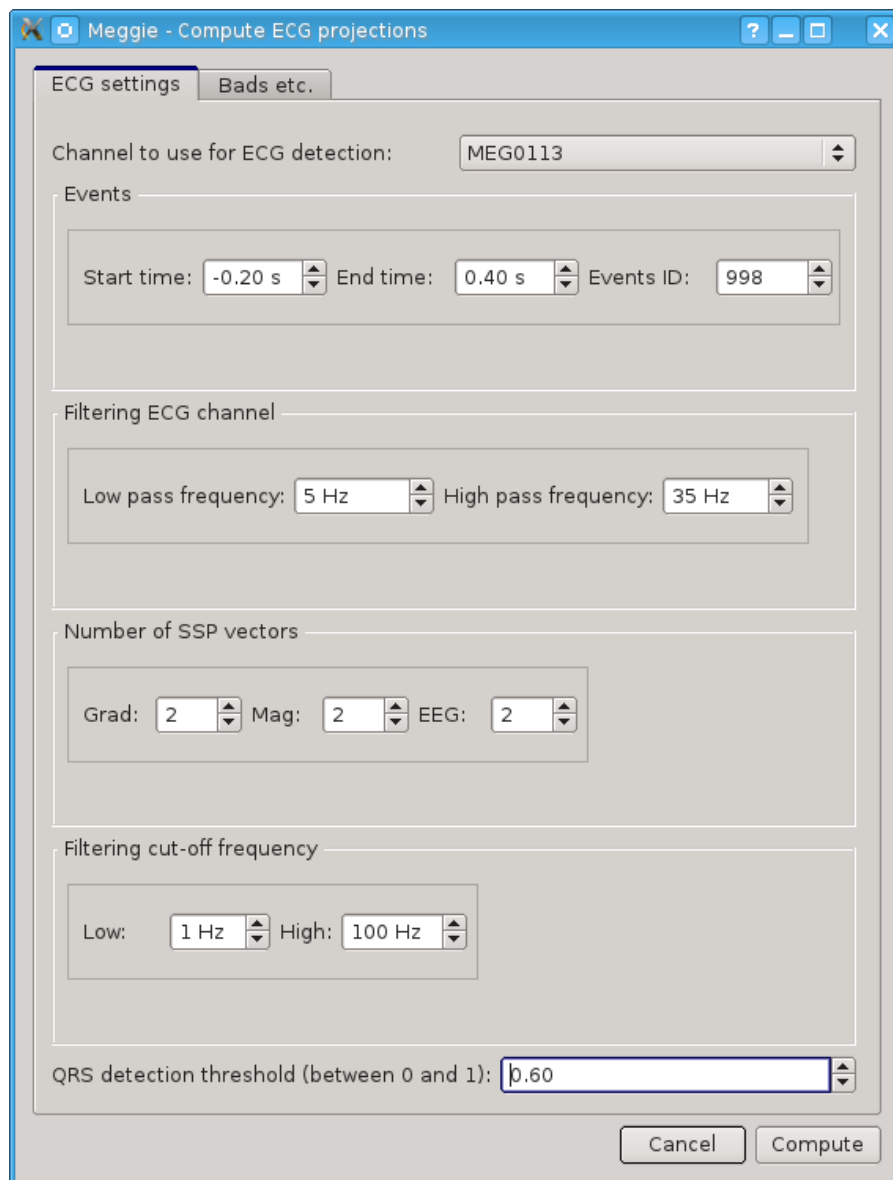


Figure 6: ECG projections dialog with *ECG settings* tab active.

The following options on first tab of the dialog may need clarification:

- *Start time*, *End time* and *Event ID*: Time before and after the events in seconds, and ID to be used for these events.
- *Number of SSP vectors*, separately for gradiometers, magnetometers and EEG.

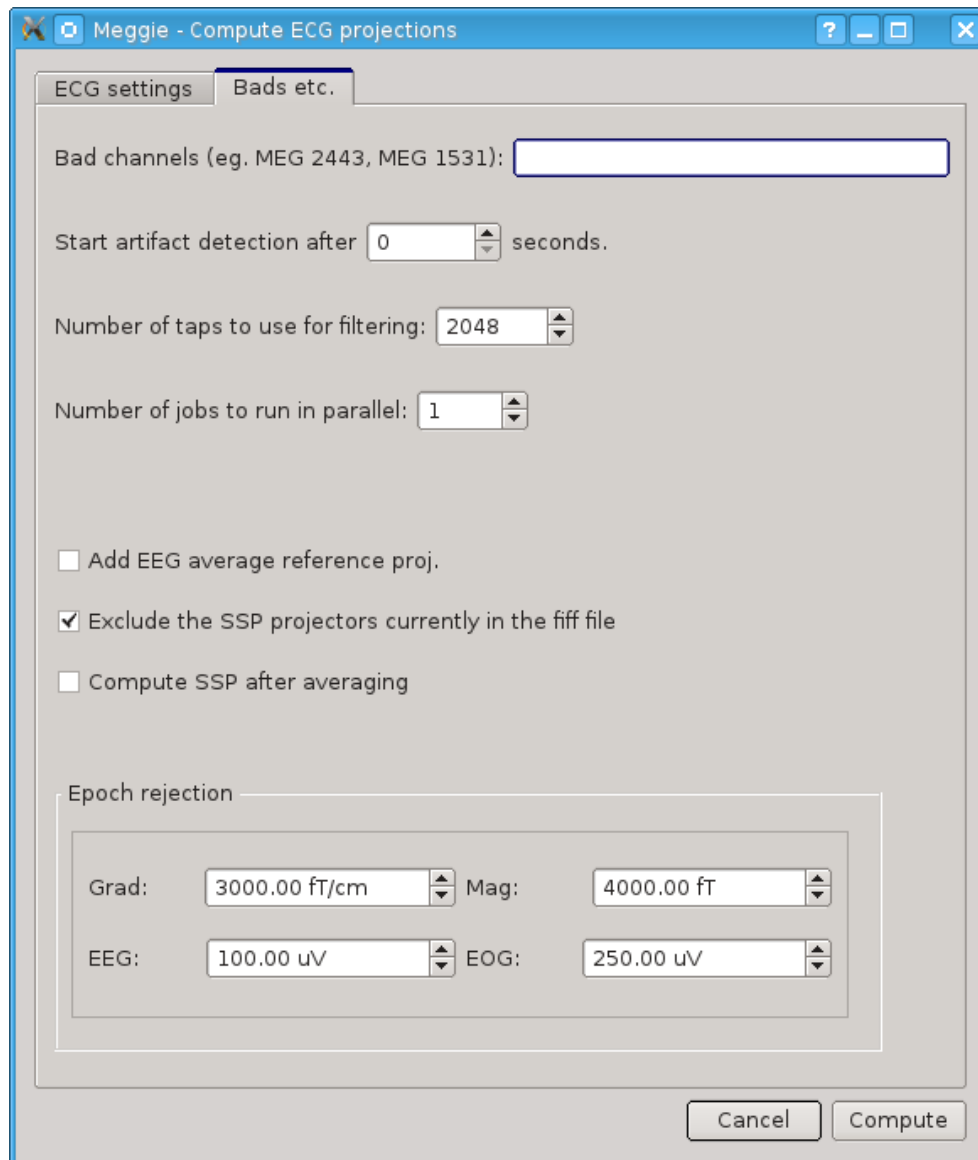


Figure 7: ECG projections dialog with other settings tab active.

The tab in Figure 7 has the following non-self-evident options:

- *Number of jobs to run in parallel*: a higher number of jobs allows the application to use multiple processor cores for computing projections. This, in turn, allows faster computation, but may prevent the use of computer for other tasks during the computation. It is not useful to exceed the number of processor cores in the CPU.
- *Epoch rejection* parameters: If the peak-to-peak amplitude of a channel during an epoch exceeds the values given in these fields, the epoch will not be included in the epoch collection.

4.6.3 Calculate EOG Projections

The button opens the dialog for calculating EOG-projections for the raw file. The dialog is similar to *Calculate ECG projections* dialog, except for the different default values and no combobox for choosing the EOG channel (it will be chosen automatically based on the metadata in the fiff file). The MNE method used is `mne.preprocessing.compute_proj_eog`.

4.6.4 Apply ECG Projections and Apply EOG Projections

The buttons apply ECG Projections or EOG Projections calculated (see Sections 4.6.2 and 4.6.3, respectively), to the current working. The MNE method used is `mne.fiff.Raw.add_proj`.

4.6.5 Magnitude Spectrum

The *Magnitude spectrum* button opens a dialog, which allows choosing a channel to show the magnitude spectrum of (see Figure 8).



Figure 8: Magnitude Spectrum dialog

4.7 Epoching Tab

Epoching tab (see Figure 9) includes a list of epoch collections created by picking events from the raw file. The actual creation of epoch collections is initiated by clicking *Create an epoch collection* button. The epoch collection list created in the tab will also be shown in *Averaging* and *TFR* tabs (see Figures 12 and 14).

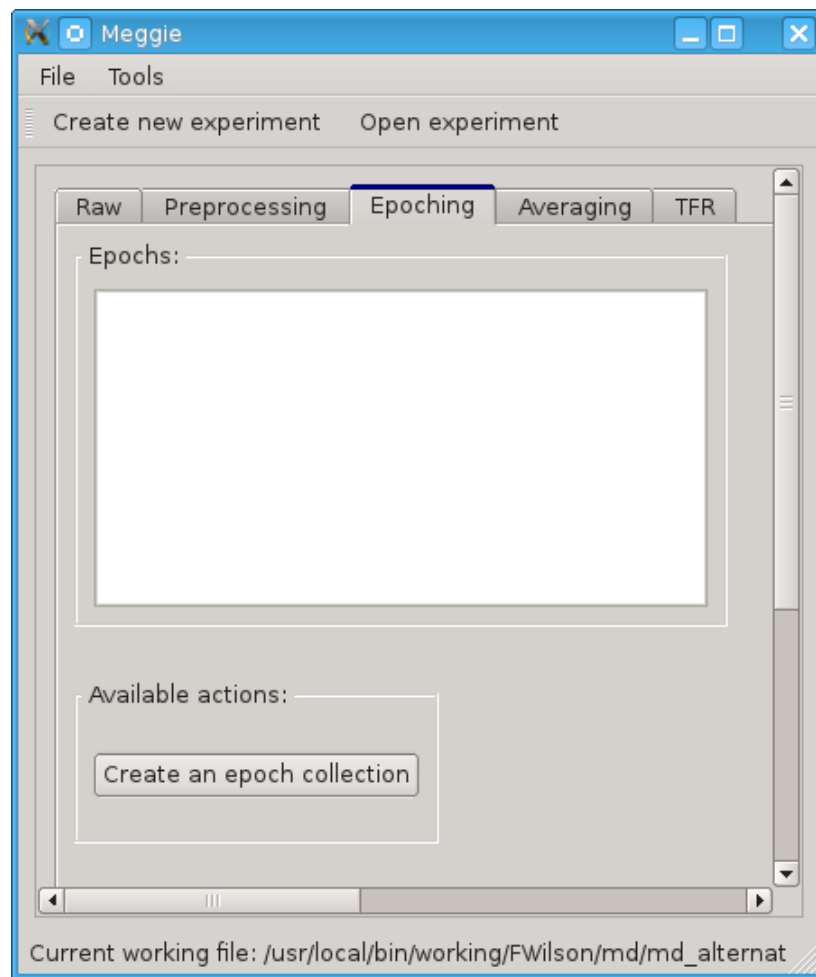


Figure 9: Main window with epoching tab active.

4.7.1 Event Selection Dialog

After clicking *Create an epoch collection* button (see Figure 9), the user is presented with the dialog (see Figure 10) allowing the picking of events from the raw file by event ID.

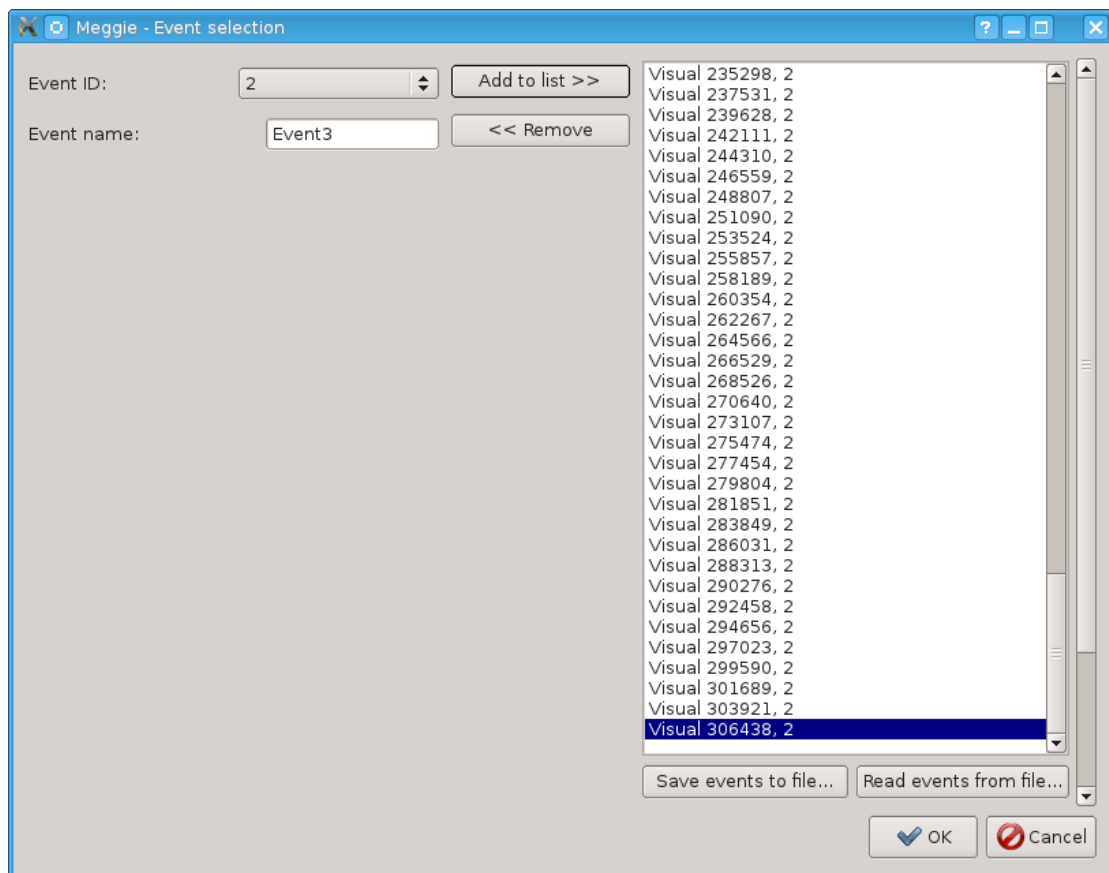
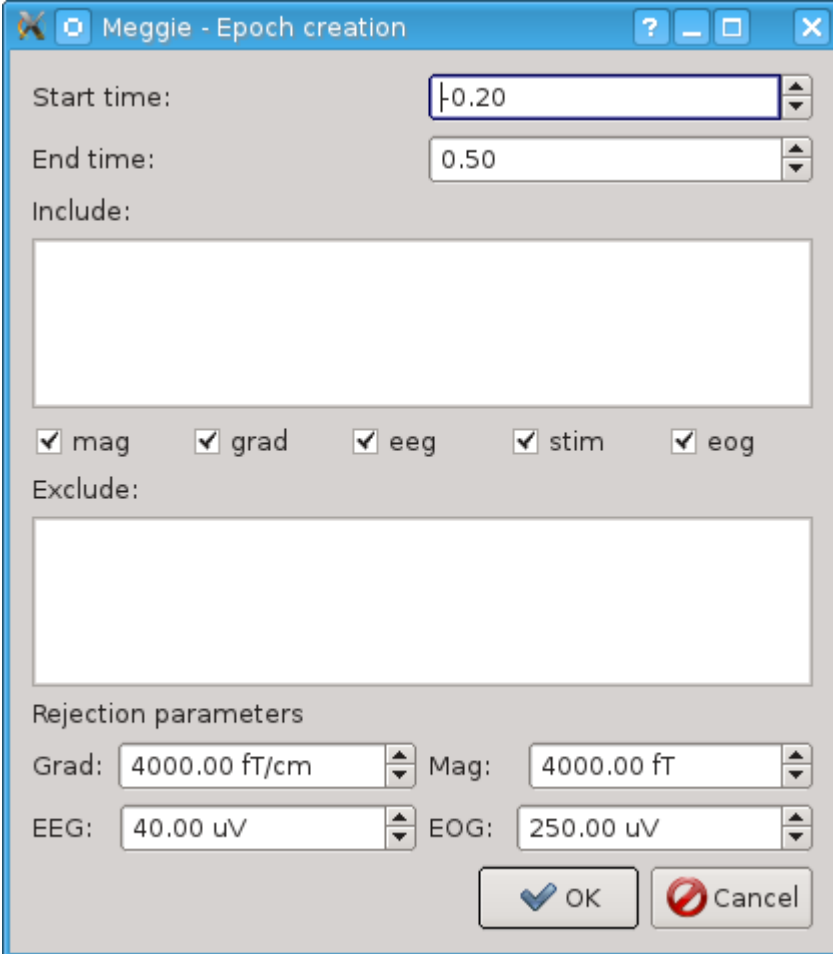


Figure 10: Event selection dialog.

Add to list button adds all events with the selected event ID to the list on the right, whereas *Remove* button removes the selected event from the list. *Save events to file* button saves the list into the subject directory in xls format, and *Read events from file* button reads an xls formatted file into the list (after clearing the list first). This allows a more fine-grained editing of the list via editors capable of editing xls-formatted files.

4.7.2 Epoch Creation Dialog

After clicking *OK* button in the event selection dialog (see Figure 10), the user is presented with the *Epoch creation* dialog (see Figure 11). The dialog allows the actual forming of epochs from the selected events in the raw file.



Meggie - Epoch creation

Start time: 0.20

End time: 0.50

Include:

☒ mag ☒ grad ☒ eeg ☒ stim ☒ eog

Exclude:

Rejection parameters

Grad: 4000.00 fT/cm Mag: 4000.00 fT

EEG: 40.00 uV EOG: 250.00 uV

OK Cancel

Figure 11: Epoch creation dialog.

The dialog has the following fields:

- *Start time* and *end time* define the time window for the epochs in milliseconds before the events (start time) and after them (end time).
- *Include* can be used to list the channels to be included, separated by commas, in addition to the channels chosen with the include checkboxes (see below). It could be used to choose single channels from the channel types not included by the checkboxes. Example of input: MEG1451, EEG006
- *Include* checkboxes (*mag*, *grad*, *eeg*, *stim*, *eog*): checking these includes all the channels of checked types in the epoching, and excludes all the channels not checked.
- *Exclude* can be used to list the channel types and numbers to exclude, separated by commas, exactly in the same way as in *Include* list. It could be used to exclude specific channels from the channels types chosen by checking *Include* checkboxes.

- *Rejection parameters.* If the peak-to-peak amplitude of a channel during an epoch exceeds the values given in these fields, the epoch will not be included in the epoch collection.

4.8 Averaging Tab

Averaging tab (see Figure 12) includes the epoch collection list created in the epoching tab (see Chapter 4.7) and the single button *Average selected epoch collection*. Clicking the button averages the selected epoch collection and visualizes the result as a 2D sensor topography view (see Figure 13). In the sensor topography view, clicking on a rectangle corresponding to a single channel shows a more detailed view of the averaged data for the channel clicked. The MNE method used for the topography is `mne.viz.plot_topo`.

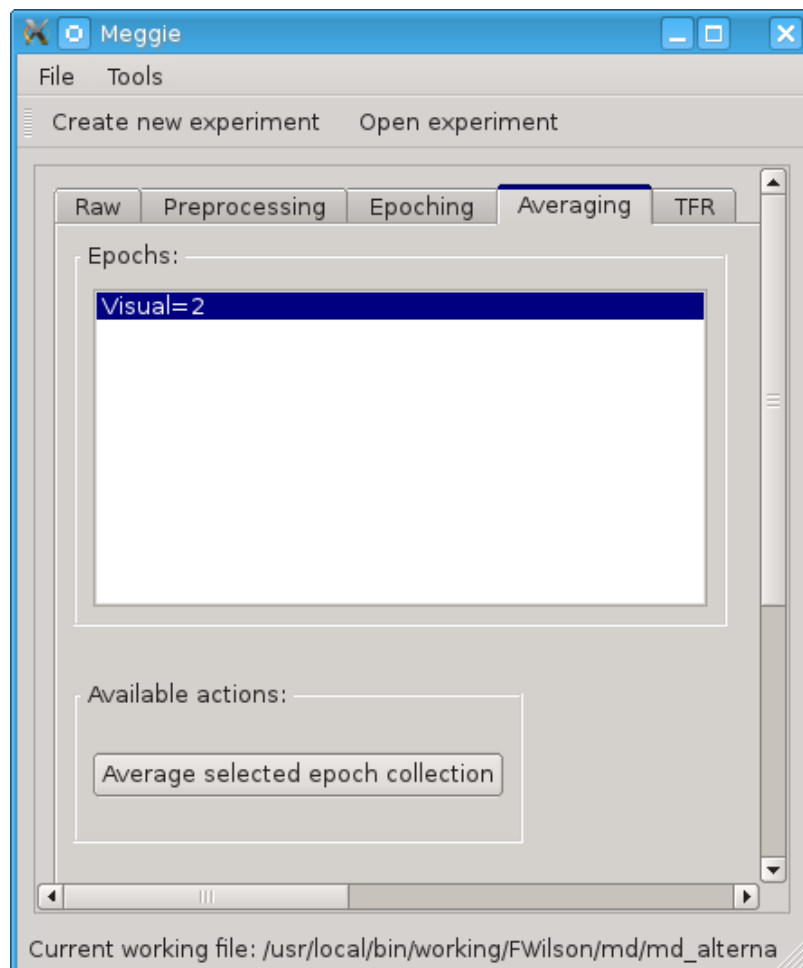


Figure 12: Main window with Averaging tab active.

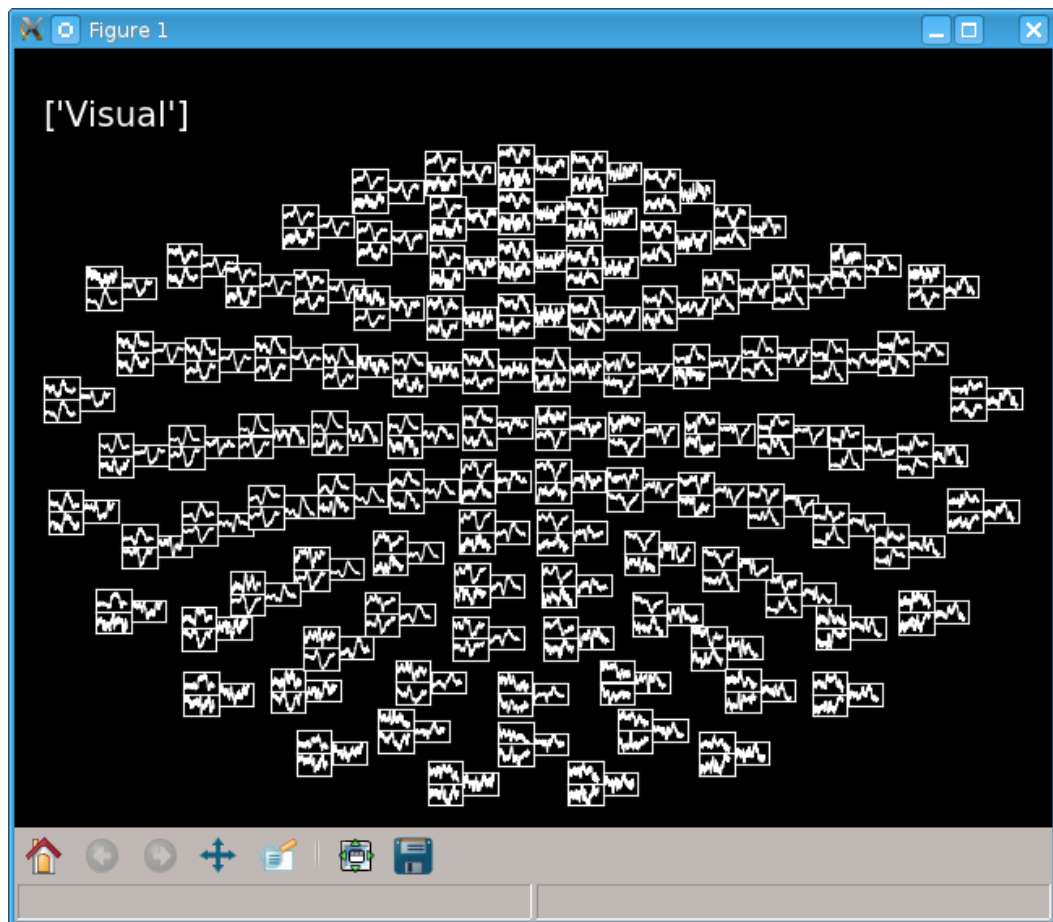


Figure 13: Average topology view.

4.9 TFR Tab

TFR tab (see Figure 14) allows visualization of epochs as Time Frequency Representation. Visualizing TFR is possible either as a single channel view with *TFR visualization* button (see Section 4.9.1) or as a 2D topography view with *TFR topology visualization* button (see Section 4.9.2).

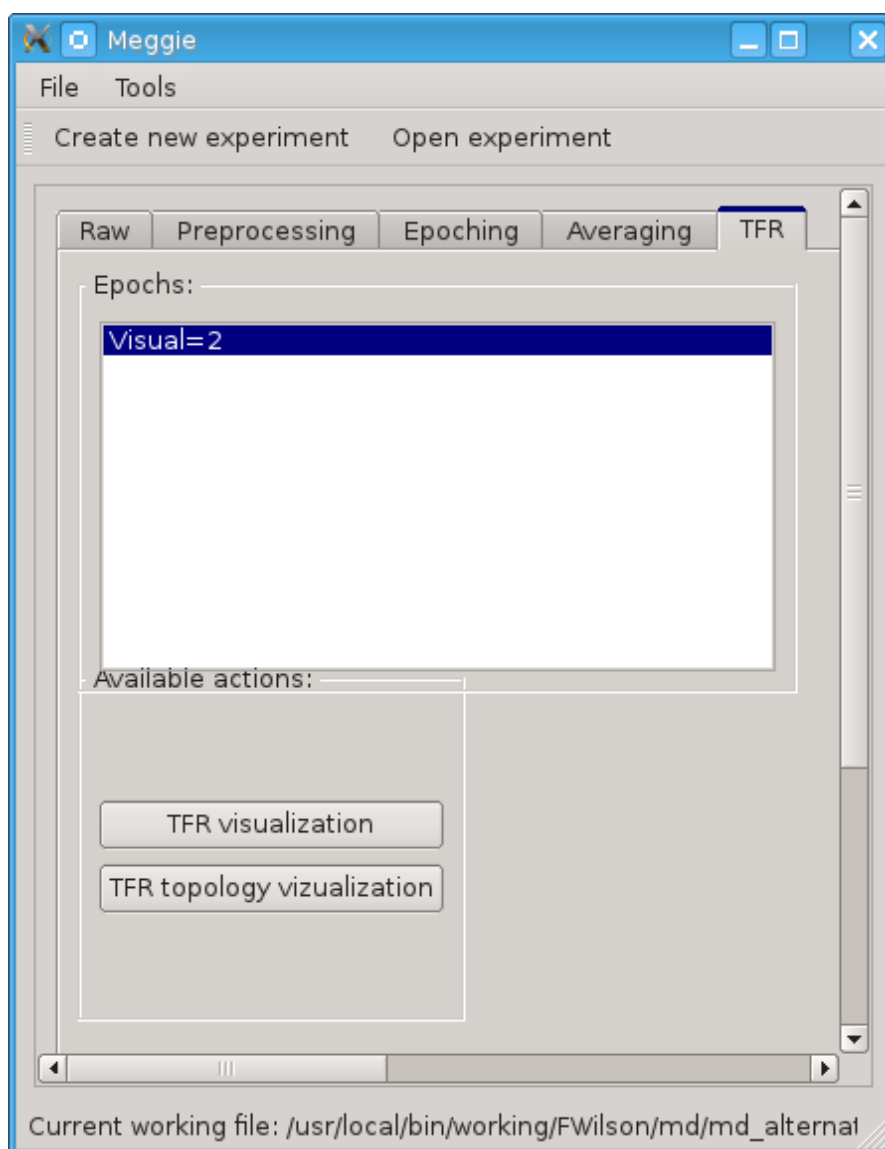


Figure 14: Main window with TFR tab active.

4.9.1 TFR Visualization

Before clicking *TFR visualization* button (see Figure 14), you have to select an epoch collection to be visualized from *Epochs* list. Clicking the button causes *TFR from epochs* dialog (see Figure 15) to appear.

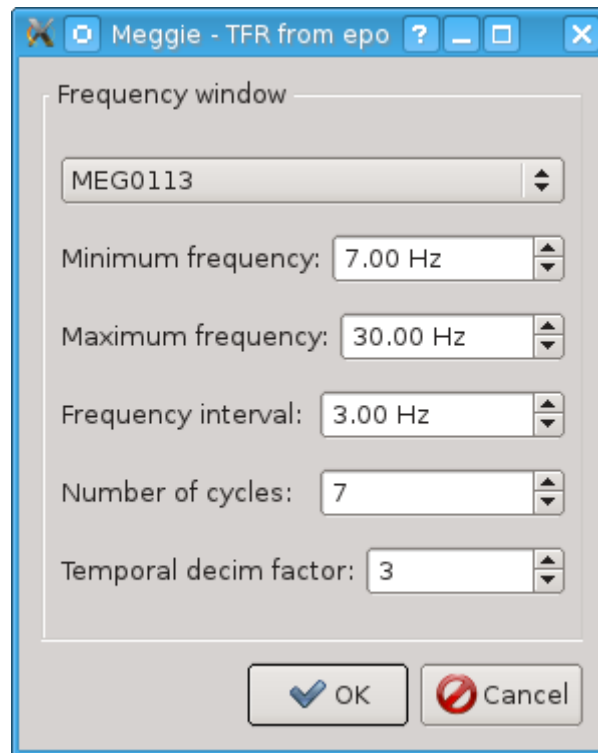


Figure 15: TFR dialog.

The dialog has the following fields:

- The combobox on the top allows selection of the channel to be visualized.
- *Minimum frequency* and *Maximum frequency* define the frequency window of interest.
- *Frequency interval* and *Number of cycles* define the interval and cycles, respectively, within the frequency window of interest.
- *Temporal decim factor* defines the temporal decimation factor for the induced power and phase locking value view (see Figure 16).

Clicking *OK* will open a dialog (see Figure 16) showing the phase locking and induced power TFR views for the selected channel, with a graph for evoked response at the top. The visualization implementation closely follows the MNE-Python script `plot_time_frequency.py`[8].

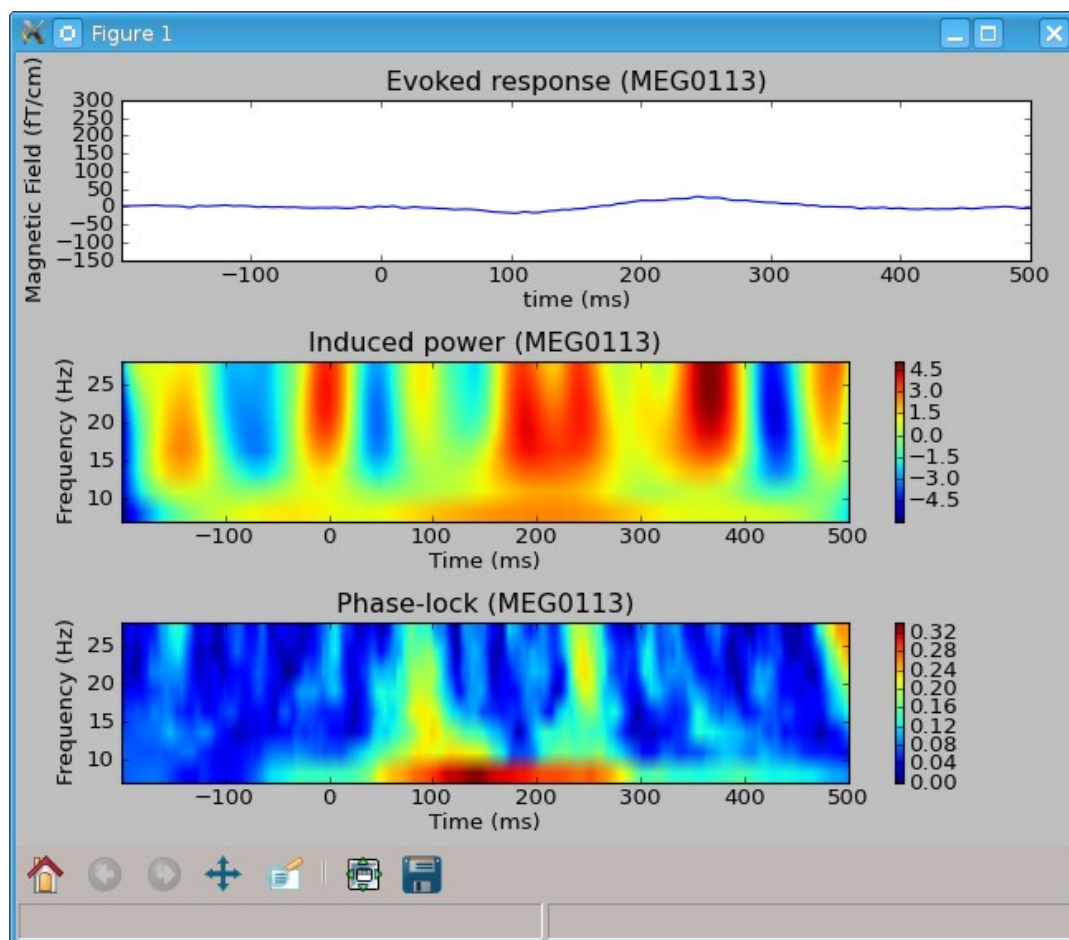


Figure 16: TFR view for a single channel.

4.9.2 TFR Topology Visualization

Clicking *TFR topology visualization* button in *TRF* tab (see Figure 14) results in 2D topology view similar to that of the *Averaging* tab. The view only shows either induced power or phase locking value for each, depending on the selection (see *Topography type* in Figure 17). The dialog also has options for baseline rescaling and correction. The input fields in the dialog are otherwise the same as in the single channel TFR view.

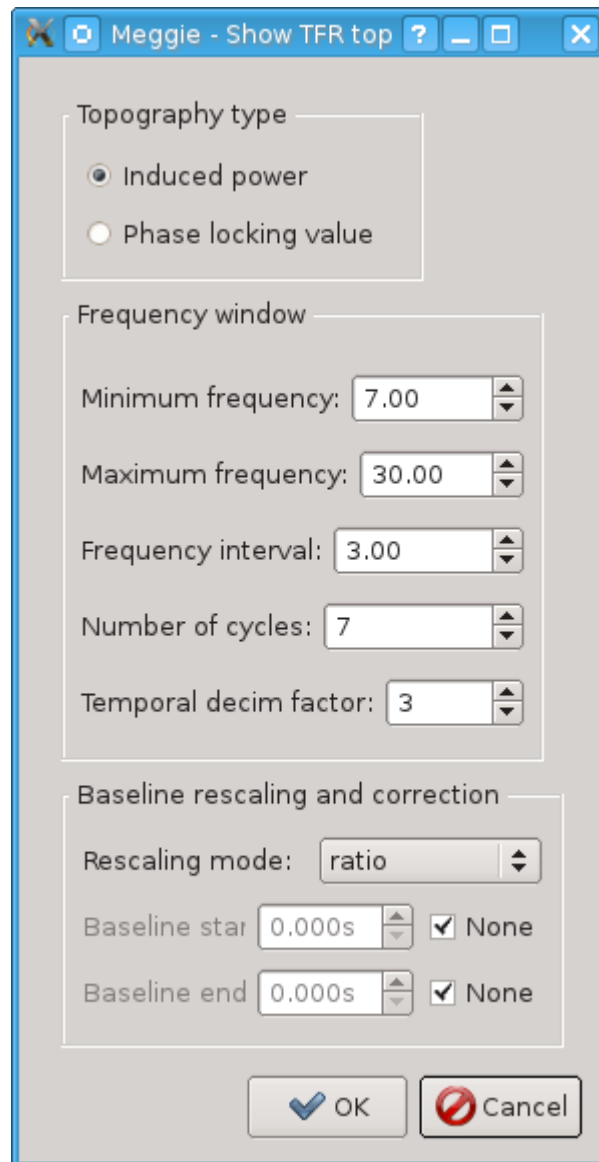


Figure 17: TRF topology dialog.

In the actual visualization view (see Figure 18), clicking on a rectangle corresponding to a single channel shows a more detailed view of the TFR data for the channel clicked. The visualization implementation closely follows the MNE-Python script `plot_tfr_topography.py`[9].

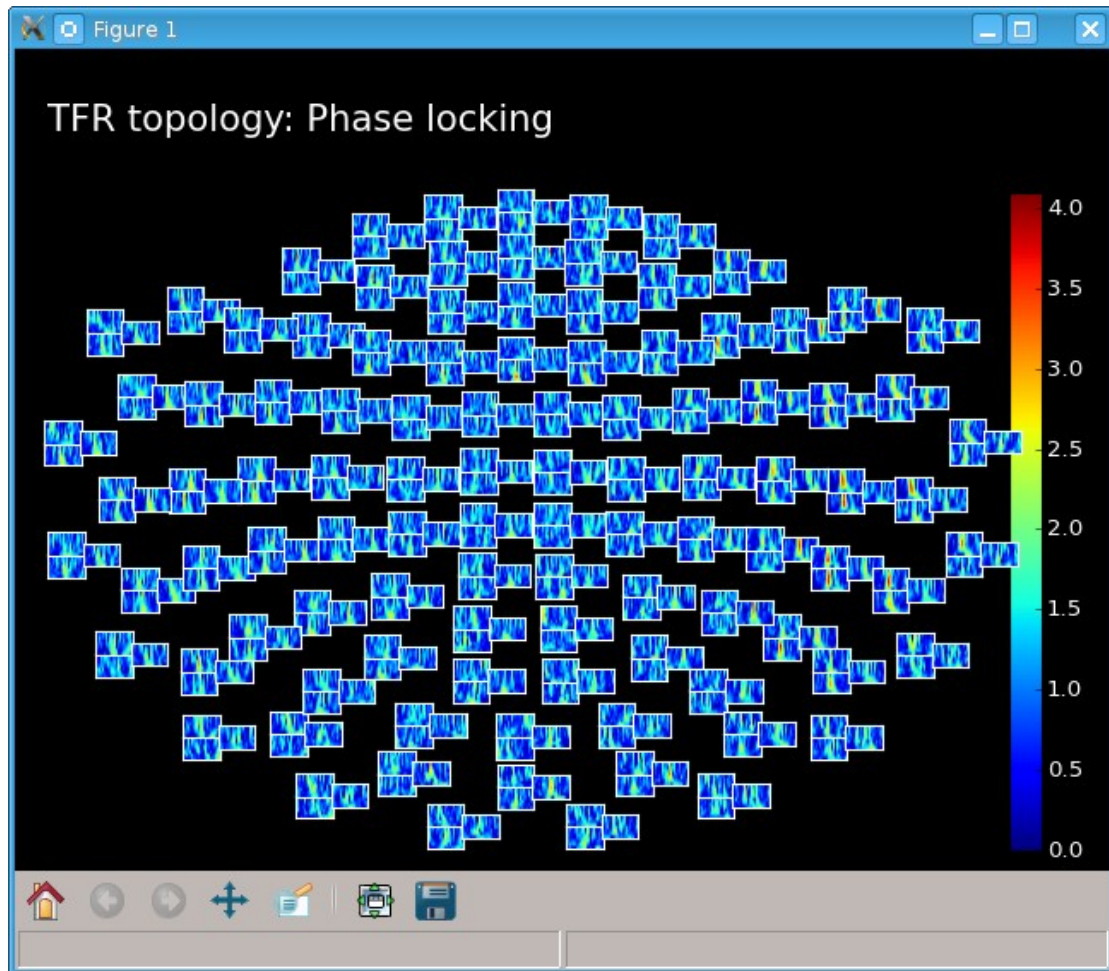


Figure 18: TFR topology view (phase locking).

5 General Application Structure

Meggie is a locally installable application with no functionality requiring network access. It also needs no specialized hardware to run. Meggie has been tested to run on Fedora 16, but may work with other versions of Fedora or other Linux distributions.

The chapter describes the different components of Meggie and how these components relate to each other.

Meggie uses many common Python libraries and also Python libraries for scientific computing, i.e. Scipy and NumPy. It also has a hard requirement on MNE-Python libraries, which take care of the actual domain related scientific computing. The external libraries and programs required for some functionality include non-Python MNE (for `mne_browse_raw`) and MaxFilter (for basic preprocessing). The user interface library used by Meggie is PyQt4.

5.1 Formats, Components and Software

The section describes file formats and external dependencies of the application.

EPD	i.e. Enthought Python Distribution, is a tailored Python distribution for scientific computing, released by Enthought Inc. It includes libraries for data analysis and visualization. It was recently renamed to Enthought Canopy.
Fiff	is a binary data format produced by MEG equipment, used for storing MEG data. Meggie reads and writes files in Fiff format via MNE-Python.
Matplotlib	is a plotting library for Python and the NumPy extension of Python. It is used by Meggie via the state machine based pylab interface.
MaxFilter	is a commercial software by Elekta Neuromag, a manufacturer of MEG equipment. It is designed for preprocessing of MEG signal data and includes SSS, tSSS and MC filtering capabilities. MaxFilter is written in C.
MNE	is an open source MNE software package, hosted at the Martinos Center for Biomedical Imaging at Harvard-MIT. It is used for analyzing and visualizing MEG and EEG data. It is written in C.
MNE-Python	is the Python module collection of MNE. It includes a large part of the functionality included in MNE, rewritten in Python.

Xlrd is a Python library for reading files in a simple spreadsheet format. It is used by Meggie for saving event lists to disk (see Figure 10 on page 15)

Xlwt is a Python library for writing files in a simple spreadsheet format. It is used by Meggie for reading events lists from disk (see Figure 10 on page 16).

5.2 Class Structure

A simplified class structure of Meggie is presented in Figure 19. The figure shows the relationships between the essential interface and logic classes of Meggie as well as external libraries. For a more detailed description, please see the Meggie Class Documentation [4].

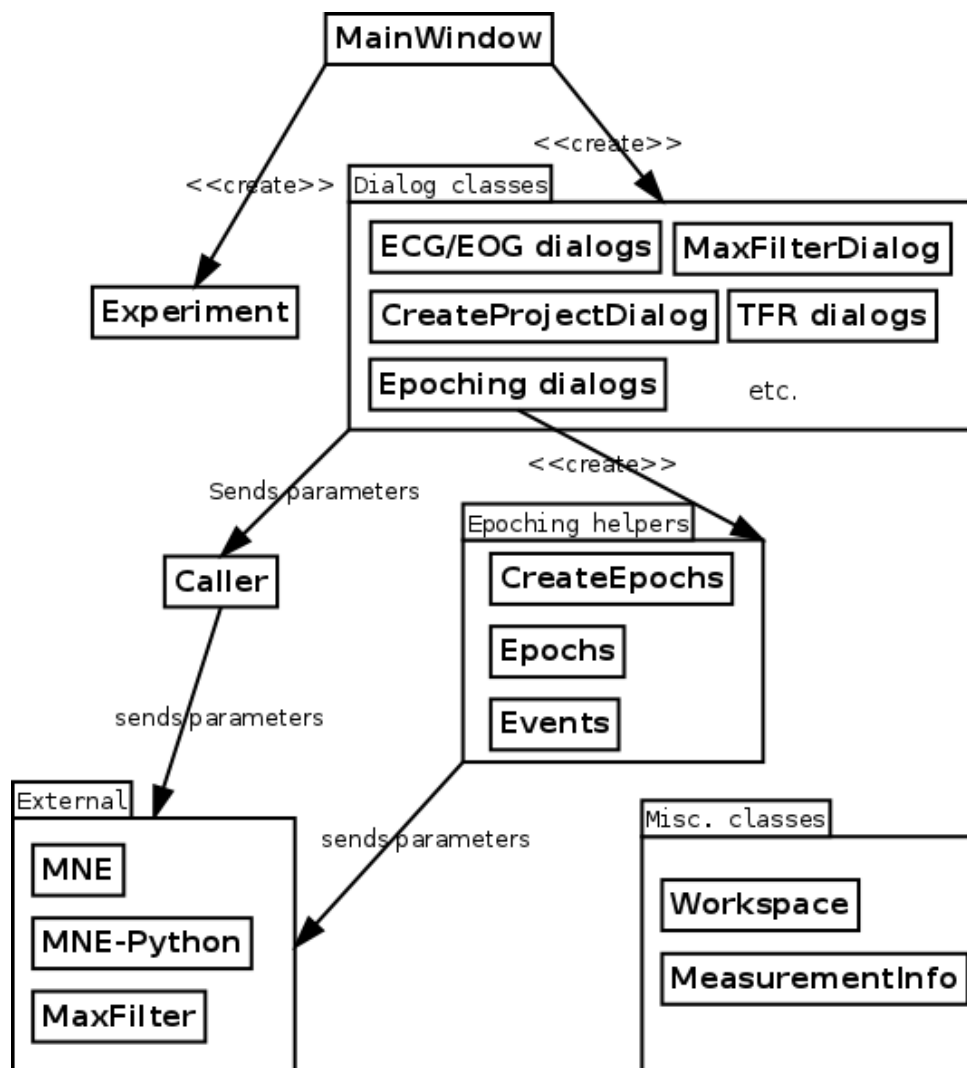


Figure 19: A simplified class structure of Meggie.

`MainWindow` class handles the main user interface of the application. It creates all dialog classes when needed and holds a permanent reference to the current `Experiment` instance presented by the user interface.

`Experiment` class stores the state and the information related to the experiment presented via the main window. It also takes care of saving and loading its state to and from disk, in addition to saving the previously used parameters of the dialog classes for future use.

The **dialog classes** are created by `MainWindow` class when the user clicks the corresponding buttons. They allow the user input of values for all the commands passed to the external libraries and applications. Each dialog is actually split into two classes in order to achieve basic separation between the user interface and the application logic. The application logic class resides in a module ending with `Main`, whereas the module ending with `Ui` includes the user interface definition class. The latter is generated by Qt Designer and PyQt tools and should not be edited by hand.

`Caller` class stands between the dialog classes and the external libraries. The dialog classes hand their values to `Caller` as dictionaries, whereas `Caller` parses the dictionaries and calls the backends with the parsed parameters. `Caller` also reports to `Experiment` whether or not the procedure was carried out successfully.

The **epoching classes** pick events from the raw file and create epochs based on the conditions passed to them as arguments. Presently, the epoching classes bypass the `Caller`, directly calling the external libraries in order to average epochs and show the averaged data as a 2D topology.

The **external libraries** take care of the actual computational heavy lifting needed by the application. They are also the main source of the exceptions of the application. MNE-Python, in particular, has very user friendly exception messages that are passed directly to the user interface.

The **miscellaneous classes** include methods for separate parts or functionalities of the user interface and application logic. They include classes for validating input and managing workspace, for example.

5.3 File and Data Formats

The Meggie application uses a single custom file format, signified with the filename extension `param`. The files in the format are created by `Caller` class, and contain information about the values received from the dialogs. The files are used for storing the previous values of the input fields of the dialogs, in order to make it easier for the user to call external applications and methods with slightly altered values.

The file format consists of four parts:

- 1) The first line includes the path to the input file of the method, for example `/usr/local/bin/working/test/md/md_alternateR_sss.fif`

2) The second line includes the path to the output file generated by the method, for example `/usr/local/bin/working/test/md/md_alternateR_sss_ecg_proj.fif`

3) The third line includes the command used for invoking the method, for example `mne.preprocessing.compute_proj_ecg`

4) The rest of the lines include the parameters and their values used for calling the method. Each line is a single parameter-value pair, separated by a comma, for example

`ecg-h-freq,35`

`qrs,0.6`

`rej-grad,3000.0`

6 Programming and Testing Practices

The programming in Hoksotin was closely intermingled with designing of the user interface and planning of the application structure in general. There was a distinct programming phase in the latter half of the project, while earlier the design, planning and programming activities took place at the same time. The testing (see Section 6.5) was quite informal and mostly took place during the latter half of the development phase.

Hoksotin project set out to follow the Python programming guidelines right from the beginning. Code readability was an important goal during the programming, and comments were mostly written at the same time as the methods and classes they refer to.

The source code of the project was reviewed twice by the technical supervisor Tuomas Puoliväli, who gave feedback on the questionable parts of the code. They mostly concerned the exception handling, the use of Python properties and the lack of comments. These parts were subsequently rewritten or refactored in order to address the issues he had brought up. Addressing some of the issues was deemed a matter of future development (see Chapter 8).

WWW sources were pivotal in overcoming many problems. Python on-line documentation [10] was referred to frequently, while a basic PyQt tutorial [11] proved useful in the beginning of the project. A popular programming forum Stack Overflow [12] was browsed in cases where the standard Python documentation was found lacking in examples.

6.1 Formatting, Naming and Commenting Practices

Readability was the foremost principle in formatting, naming and commenting the source code. Hoksotin group followed the standard formatting practices [13] used in Python programming, adhering strictly to the recommendations about line lengths and blank lines.

The names of the methods and variables were designed to be as self-evident as possible. Due to the complexity of the domain, however, detailed comments were still written for all methods, especially for those that call backends with numerous non-self-evident parameters.

The multipart names in the actual code were consistently written in `snake_case`, whereas the module names were written in `CamelCase`. The class names were started with uppercase letters, whereas the methods were written wholly in lowercase. All source code, including comments, was written in English.

6.2 Source Code Example

The following example is taken from the file `measurementInfo.py`. It illustrates the formatting, naming and commenting practices used in Hoksotin project.

```
# coding: latin1
"""
Created on Mar 6, 2013

@author: Kari Aliranta, Jaakko Leppakangas
Contains the MeasurementInfo class used for collecting information
from MEG measurement raw files.
"""
import mne

import datetime
import re

class MeasurementInfo(object):
    """
    A class for collecting information from MEG-measurement
    raw files.
    """

    def __init__(self, raw):
        """
        Constructor

        Keyword arguments:
        raw -- Raw object
        Raises a TypeError if the raw object is not of type
        mne.fiff.Raw.
        """
        if isinstance(raw, mne.fiff.raw.Raw):
            self._raw = raw
            self._info = dict(raw.info)
        else:
            raise TypeError('Not a Raw object.')

    @property
    def high_pass(self):
        """
        Returns the online high pass cutoff frequency in Hz.
        Raises an exception if the field highpass does not
        exist.
        """
        if self._info.get('highpass') is None:
            raise Exception('Field highpass does not exist.')
        else:
            return round(self._info.get('highpass'), 2)
```

6.3 Grouping Practices

The source code was divided into different files according a rough separation of user interface and application logic. The user interface code lies within `ui` package, whereas the non-ui-related application code lies withing `code` package. Both packages have subpackages named according to the steps of processing and analysis (see tabs in Figure 4), or simply `general` if related to the general logic of the application. The subpackages, in turn, include the actual code-containing modules related to the corresponding step.

The program code related to each window and dialog is split into modules as described in Section 5.2 on page 26.

6.4 Development Platform

The application was developed on Fedora 16, on Gnome and KDE desktop environments. Some preliminary testing with Ubuntu 12.04 was carried out, but no development took place on other Linux distributions or operating systems.

The main development platform used was **Eclipse IDE** with **PyDev** plugin. Linux text editors, like Kate or Gedit, were used rarely.

The source code uses the latin-1 encoding. **Version control** was handled by YouSource, which uses Git.

6.5 Testing Practices and Results

The testing of the application was carried out continuously during the development, yet most effort was concentrated on the latter part of the project. The tests mostly included usability testing and manual integration testing, carried out by the developers on their workstations and, to a lesser extent, the instructors and the customer on their own computers. Some unit tests were also written, but the program code they stress is not currently in use in the application.

The customer provided the group with several different kinds of files to use as raw files for the application. Some of the files included missing info fields and nonstandard field content. Raw files containing continuous data turned out to be especially troublesome and remain a focal point for the future development.

The group also carried out some manual fuzz testing on the input values of dialogs. It uncovered some bugs and anomalies in value checking and error reporting in the process.

For the usability testing the group observed the customer using the application and documented the problems and bugs observed. The testing also served to validate the requirements specified for the application. The observations were recorded for the future development (see Chapter 8).

The main **results of the testing** relate to the following areas:

- User feedback about the state of the application.
- Unexpected behavior when the user breaks the intended sequence of commands.
- Input validation, e.g.. checking for non-ascii characters and whitespaces.

The most important testing results are listed as essential bugs in Section 8.1.

7 Realization of Objectives

The chapter describes the objectives set to Meggie and how well they were met. Hoksotin group concentrated their development effort on preprocessing steps of the MEG measurement data. Also, some parts of analysis and visualization were implemented. Noticable features that were not implemented includes source space analysis, visualization and the collection of the results of the analysis process.

7.1 Realization of Requirements

Software Requirements Specification [2] includes 81 requirements. Due to the limited amount of workhours Hoksotin group was able to implement 43 of these requirements. The group had little time to test the features thoroughly, since the customer wanted to complete a certain analysis path in a typical case, rather than to be sure that the more rarely used features work properly.

Requirements concerning preprocessing were almost all implemented, excluding batch processing and progress bars. Requirements related to the commercial MaxFilter program formed a special case: most of them were marked as implemented, but were impossible to test without the actual program, which was unavailable during the development.

None of the requirements related to source space analysis were implemented.

The requirements were given priorities Obligatory (1), Important (2) and If time permits (3). 30 out of 34 requirements of priority one were fully implemented, and the remaining four included two partially implemented and two unimplemented requirements. 13 out of 42 requirements of priority two were implemented, and the remaining 29 included 5 partially implemented and 24 unimplemented requirements. None of the priority three requirements were implemented. All of the priority one and two requirements are essential and should be taken into consideration in the future development.

7.2 Unsatisfactory Solutions in the Implementation

Presently, it is not possible to easily save all the details of the files created by application. These details include statistical and time series information related to the processed files. This is a problem should the user want to use results discovered with the application for scientific publications.

Another unsatisfactory solution is related to saving, removing and editing of epoch collections. The epoch collections are currently not saved along with the experiment, and the epoch collection widget currently allows neither removal nor editing of the created collections.

Additionally, some user interface elements have incomplete value checking and reporting on incorrect values. Some elements should also be named more properly and others should be altogether removed.

The structure of the program code is also in need of some refactoring. There are functionalities that should be separated into proper classes of their own, such as saving files, handling file paths and collecting parameters. Examples of the classes in need of refactoring include `Experiment` class and the `Main` halves of the user interface classes.

Additionally, technical supervisor Tuomas Puoliväli noticed some exceptions with missing feedback to the user [14]. Supervisor Jukka-Pekka Santanen also listed user interface shortcomings based on his testing of the application [15]. Many of issues listed by Puoliväli and Santanen have already been addressed.

7.3 Challenges in the Implementation

The biggest challenge during the development was the initial lack of domain knowledge. Brain signal analysis wasn't an area of expertise for any of the group members, and acquiring the sufficient level of knowledge required a lot of time and effort. It was therefore not easy to come up with original solutions for the programming and user interface design problems encountered, making it hard to reach the actual implementation phase of the project. The lack of precursory graphical user interfaces to draw ideas from also exacerbated the problem.

The lack of domain knowledge also resulted in late specification of requirements. This, in turn, led to some members of the group finding it hard to come up with anything useful to do. Earlier meeting with the technical specialist of the customer would have helped a lot, as the specialist had the best knowledge available concerning the external modules used in the development and the brain signal analysis in general.

Further problems were also caused by the fact that none of the project members had used Python or PyQt before. Some of project members also had never used Linux. These problem, however, were anticipated and a natural part of the project.

On the infratructure side, the problems with receiving the final development computers caused a loss of some working hours, as the group had to reinstall the development environment on the new computers. On the positive side, the reinstallation forced the group to write specific installation instructions very early in the development.

7.4 Modifications During the Implementation

The most modified view of the application was the main window. The group had about 10 different design suggestions made with Qt Designer during the development of the application. Initially the group suggested that the main window should include a tree view widget, as it would have allowed an easy tracking and backtracking of the analysis process. The customer, however, disagreed on the use and usability of the backtracking feature. Eventually, the group agreed on a simple tab based design, in which a single tab represents a single step in the analysis.

The internal structure of the application was modified several times during the development as the scope of the project was redefined. The relations and roles of the central classes such as `Caller` and `Experiment` were debated in various stages of the development, resulting in severe refactoring.

Learning the specifics of Python programming language also generated some need for modifications. For example, initially the group had little knowledge about properties and encapsulation in Python. The exception handling was also almost nonexistent up until the final stages of the development. Fortunately, the technical instructor provided good instructions on both subjects and made it possible for the group to follow the recommended programming guidelines and keep the internal structure of the application sane.

8 Guide for Future Developers

The customer decided that the software developed in Hoksotin project is going to be developed after the end of the project. The chapter provides some tips to guide the future developers so that the most essential issues of the software would be addressed as soon as possible. For a more extensive look on the open issues, please see the Software Requirements Specification [2].

8.1 Essential Bugs

Meggie currently has several known bugs and some of them limit the usability of the software considerably. The following bugs should be highly prioritized in the further development:

- Epoch collections of the experiments are not saved. Presently, closing an experiment and then reloading it from disk clears the epoch collection list. On the other hand, epoch collections from previously processed experiments are not automatically deleted when creating a new experiment.
- Meggie is designed to work in Linux and it works well in Fedora, but further testing with Ubuntu uncovered some noticeable bugs. Ubuntu is a quite popular Linux distribution and it may be worthwhile to spend some time on making Meggie work on it, too.
- Computing EOG/ECG-projections currently modifies the working file. This should only happen by pressing the corresponding *Apply* button.
- The filename suffixes do not work correctly when saving the resulting file of various steps in preprocessing. Specifically, completing the same step multiple times causes the creation of new files with cumulative suffixes, while the desired behaviour should be overwriting the first file created.

8.2 Improvements to Existing Features

Before starting to improve the existing features the developers are suggested to go through the unsatisfactory implementation solutions described in Section 7.2. In addition, the following improvements to existing features should be taken into account:

- Event selection dialog (see Figure 10) should have more features for better usability. These features could include an option to add multiple event categories to the event list at once, or an option to define the directory when saving the events list to a file.

- It should be possible to choose multiple epoch collections from the epoch collections list. This would, for example, allow showing several epoch collections simultaneously in the average topology visualization view (see Figure 13).

8.3 Further Development Ideas

The application should provide more feedback about an ongoing computation process, so that the user is aware there is something sensible going on and the application simply hasn't crashed. A simple activity indicator would suffice for a start, while later developments may allow directing the output from the backends to the user interface.

There should be an option to cancel a lengthy computation gracefully should the user notice she has started it accidentally.

When completing some steps during the analysis, the user might want to add multiple raw files to be handled at the same time, with the same parameters. A separate user interface element for batch processing might be the way to implement this feature.

It should also be possible to have multiple measurement files open at the same time. This would make it easier to compare data from the different measurements.

An ability to export the results of the analysis to a specifically designed global central server could be useful. Possible advantages include an easy dissemination and comparison of research results.

8.4 Improvements to Development Practices

Section 9.1 in the Project Report [3] describes some practices that were found ineffective during the project or would have been useful if they had been used. One additional important thing that should have been done in Hoksotin project was found.

Make better use of Git, i.e. do not be afraid of manual merging. The group did not find the default merge tool very usable, resulting in some members of the group taking local backups of their local changes before pulling. The future developers should explore optional merge tools in order to feel comfortable with manual merging and co-editing of the program code.

9 Summary

Hoksotin project developed a software prototype for analyzation of MEG data for the Jyväskylä Centre for Interdisciplinary Brain Research in the University of Jyväskylä. Hoksotin did not complete all of the essential requirements set for the prototype due to the tight schedule and challenging domain of the project. Many of the features were specified too late, and the group had little time to spend on their development. Fortunately, the customer decided to continue the development of the application after the project.

The user interface of the application represents one of the four typical analysis paths of MEG data analysis. All typical paths start with the same steps, so there is a readymade base for the development of other paths, too. Additionally, the codebase has underlying functionalities for statistical analysis not implemented yet in the user interface. The application is currently not suitable for production use.

The application, called Meggie, was developed using Python 2.7 and it relies heavily on the MNE-Python libraries for its functionality. The user interface was developed using PyQt4.

The application was once reviewed by a usability expert and the source code was reviewed twice by the technical instructor of Hoksotin project. The application was preliminary tested for usability and integration. Further tests were agreed with the customer to be carried out during the future development.

10 References

- [1] MNE Developers, MNE software package, available at URL: <http://martinos.org/mne/>, cited at 2013-06-21.
- [2] Kari Aliranta, Jaakko Leppäkangas, Janne Pesonen and Atte Rautio, “Hoksotin-sovellusprojekti, Vaatimusmäärittely” (Hoksotin Application Project, Software Requirements Specification), University of Jyväskylä, Department of Mathematical Information Technology, 2013.
- [3] Kari Aliranta, Jaakko Leppäkangas, Janne Pesonen and Atte Rautio, “Hoksotin-sovellusprojekti, Projektiraportti” (Hoksotin Application Project, Project Report), University of Jyväskylä, Department of Mathematical Information Technology, 2013.
- [4] Kari Aliranta, Jaakko Leppäkangas, Janne Pesonen and Atte Rautio, “Meggie Class Documentation”, University of Jyväskylä, Department of Mathematical Information Technology, 2013.
- [5] Kari Aliranta, Jaakko Leppäkangas, Janne Pesonen and Atte Rautio, “Hoksotin-sovellusprojekti, Installation Guide”, University of Jyväskylä, Department of Mathematical Information Technology, 2013.
- [6] MNE Developers, MNE Python reference guide, available at URL http://martinos.org/mne/python_reference.html, cited at 2013-06-21.
- [7] “MaxFilter User Guide”, Elekta Oy, Helsinki, Finland, 2010.
- [8] MNE with Python, Examples, “Time frequency: Induced power and inter-trial phase-lock”, available at URL: http://martinos.org/mne/auto_examples/time_frequency/plot_time_frequency.html, cited at 2013-06-25.
- [9] MNE with Python, Examples, “Plot time-frequency representations on topographies for MEG sensors”, available at URL: http://martinos.org/mne/auto_examples/time_frequency/plot_tfr_topography.html, cited at 2013-06-25.
- [10] Python Documentation, available at URL <http://www.python.org/doc/>, cited at 2013-06-21.
- [11] Roberto Alsina, “PyQt by Example”, available at URL <http://lateral.netmanagers.com.ar/stories/BBS47.html>, cited at 2013-06-21.
- [12] Stack Overflow main page, available at URL <http://stackoverflow.com/>, cited at 2013-06-24.
- [13] Guido van Rossum and Barry Warsaw, “Style Guide for Python Code”, available at URL: <http://www.python.org/dev/peps/pep-0008/>, cited at 2013-06-25.

-
- [14] Tuomas Puoliväli, “Virhetilanteita” (Error Conditions), available at Hoksotin mail archive at URL: https://korppi.jyu.fi/kotka/servlet/list-archive/hoksotin_opetus/0045.html, University of Jyväskylä, Department of Mathematical Information Technology, 2013.
- [15] Jukka-Pekka Santanen, “Huomioita sovelluksesta koekäytön perusteella” (Notes About the Application Based on a Test Run), available at Hoksotin mail archive at URL: <https://korppi.jyu.fi/kotka/servlet/list-archive/hoksotin/0118.html>, University of Jyväskylä, Department of Mathematical Information Technology, 2013.