**Boston EEG Automated Processing Pipeline (BEAPP) User Guide**

**Version 4.1 Beta**

**September 2017**

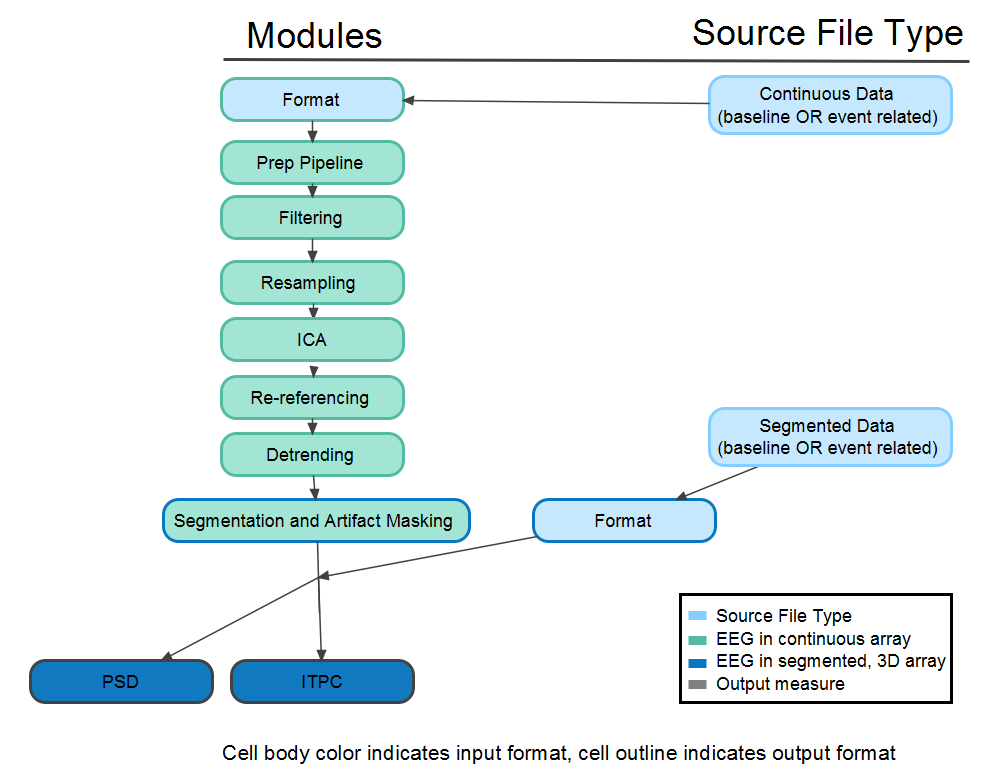
**What is BEAPP?**

The Boston EEG Automated Processing Pipeline (BEAPP) is a modular, Matlab-based software designed to facilitate automated, flexible batch processing of baseline and event-related EEG files in datasets with mixed acquisition formats.

Rather than prescribing a specified set of EEG processing steps, BEAPP allows users to choose from a menu of options. Each option can be turned on or off, and options turned “on” can be tailored to fit the user’s needs. BEAPP currently provides options for the following user-controlled modules:

1. [PREP Pipeline](http://journal.frontiersin.org/article/10.3389/fninf.2015.00016/full)
   1. Line noise removal, interpolation of bad channels, robust average referencing
2. Filtering
   1. High-pass
   2. Low-pass
   3. Notch
   4. Cleanline
3. Resampling
4. Independent Components Analysis (ICA) with optional use of [MARA](https://github.com/irenne/MARA) artifact classifier
5. HAPPE Pipeline
   1. Select 10-20 channel locations, and other channels of interest
   2. 1 Hz high-pass filter
   3. Cleanline to remove line noise
   4. Wavelet cleaning
   5. ICA with MARA
   6. Interpolate bad channels
   7. Average reference
6. Re-Referencing
   1. Laplacian (CSDLP)
   2. Average re-referencing
   3. Reference to individual or subset of electrodes
7. Detrending
   1. Mean
   2. Linear
   3. Kalman
8. Amplitude-based artifact detection
9. Segmentation
   1. Stimulus-locked (for task-related data)
   2. Non-stimulus-locked (for continuous or “resting” data)
10. Power spectral decomposition (PSD)
11. Inter-trial phase coherence (ITPC)

BEAPP aims to strike a balance between assuming only a basic level of MATLAB and EEG signal processing experience, while also offering a flexible menu of opportunities for more advanced users. At a minimum, no programming experience is required to use BEAPP, but basic familiarity with troubleshooting in Matlab will likely come in handy.



**Next Steps:**

BEAPP is intended to be a dynamic, rather than static, platform for EEG processing. This means that we plan to continue adding additional functionality over time, and we encourage other users to add functionality as well.

**What’s on Our Wishlist (coming soon):**

1. Formatted dataset-wide run reporting (general dataset statistics, formatted warnings in a report)
2. Reading files in directly from .bdf/.edf and .set files
3. Coherence
4. Phase lag index
5. Baseline with conditions (e.g. eyes open and eyes closed)
6. Topoplotting outputs with mixed source acquisition layouts/ number of channels
7. Phase amplitude coupling
8. GUI for user inputs
9. Ability to change the order of modules

**Start-Up Guide:**

This brief start-up guide provides a basic overview of how to use BEAPP, for a beginning user. Details on modules and settings are provided later in the guide.

1. Make sure you have the software necessary for running BEAPP:

* **Matlab (recommended 2016a or newer):** BEAPP was written in Matlab 2016a. Older versions of Matlab may or may not support certain functions used in BEAPP.
* **BEAPP:** BEAPP can be downloaded from the Github repository at \_\_\_\_\_.

1. Unzip BEAPP on your computer in the location you would like to run from.

3. Place all of the EEG files you plan to process into a single folder on your computer. Do not place them in the BEAPP directory. BEAPP can currently handle the following types of files:

* **.mat files.** These may be exported from EGI, Biosemi, ANT, or other EEG data platforms. Each .mat file must contain a variable with the (unsegmented) EEG data, in matrix format. Each row in the matrix should contain the amplitudes of the EEG tracing for a given channel across time. Each column should contain the amplitudes of the EEG tracing for a given time point across channels. The name of this variable can be whatever you typically use in your exports, but should be consistent across files. Currently, BEAPP is only able to handle continuous data from to handle .mat files; if you would like to analyze event-tagged data, you will need your files in .mff format.
* **.mff files.** These files are typically exported from NetStation (EGI).

For additional information on source file formats, see “[Running BEAPP with Different Source File Formats](#Running_BEAPP_with_Different_Formats)” later in this guide.

4. Set up tables with necessary information on each file. Templates for these tables can be found in beapp/user\_inputs, and example scripts for generating them can be found in /reference\_data/example\_scripts

* For .mat files, set up mat\_file\_info\_table.mat with the following information for each file:
  + File Name (for all the EEG files you wish to process)
  + Sampling rate
  + Net Type (See section on [BEAPP Net Library](#BEAPP_Net_Library))
  + Line noise frequency (typically either 50Hz or 60Hz, depending upon country. See section on [Line Noise Frequency](#Line_Noise_Frequency) in Formatting Specifications below.)

When you have edited this table as necessary, navigate to beapp/user\_inputs and type:

save(‘mat\_file\_info\_table.mat’,’mat\_file\_info\_table’);

* For .mff files, you have the option to set up an mff\_file\_info\_table.mat with information about event offsets and/or line noise frequency for each file. If you do not have event offsets (or if these are already accounted for in your mff file), you do not need to set up this table.

When you have edited this file as necessary, navigate to beapp/user\_inputs and type:

save(‘mff\_file\_info\_table.mat’,’mff\_file\_info\_table’);

4. Choose how your EEG data will be processed. The user inputs determine which modules to run, and the details of what will occur in those modules. *The majority of this user guide will focus on the details of these user inputs.*

* Most of the user inputs can be found here: beapp/user\_inputs/beapp\_userinputs.m. If a standard input does not apply to the processes you are running for your dataset, you can leave it set to the default values without impacting your data.
* More advanced inputs can be found here: beapp/user\_inputs/beapp\_advinputs.m. If you make any edits to the advanced inputs, be sure that in beapp\_userinputs.m you set grp\_proc\_info.beapp\_advinputs\_on=1

5. Once the above steps are complete, you’re ready to run BEAPP. Navigate to your BEAPP directory, and on the command line type:

beapp\_main

Then hit “Enter.”

6. Enjoy using BEAPP!!!

Note: It’s strongly recommended that you run a few test files through the pipeline before you begin batch processing, especially the first few times you use BEAPP. This will ensure your user settings are correct and you have a chance to adjust without needing to rerun large numbers of files unnecessarily. If you’re running .mat files, you can do this by only including a few files in mat\_file\_info\_table.mat. If you’re running .mff files, you can do this in several ways, but the easiest is to only including a few files in your source directory at first.

Note: Users wishing to convert outputs to EEGLAB format should use the beapp2eeglab and batch\_beapp2eeglab functions found in the example\_scripts folder.

**Glossary for Terms as Used in BEAPP**

**Recording Periods/ Epochs :** Subdivisions within one source file used to separate experimental paradigms or delineate breaks. In Netstation, these are called epochs. Note: Some sections of BEAPP code and documentation still use “epoch” to refer to this, but future versions will use recording period.

**Segment:** A discrete portion of EEG taken from a recording, to be used for analysis. In EEGLAB, these are called epochs.

**Task:** Experimental paradigm with its own set of stimuli, e.g oddball task.

**Events/ Event Tags:** Markers for the presentation of a stimulus or an annotation to a file.Can be added to a file by a user during preprocessing or by EPrime, Presentation, etc.

**Condition:** Variations of an event or stimulus that can occur as part of a task (for example the ‘repeating’ vs. ‘oddball’ stimuli in an oddball task).

**Cell Codes:** Number codes associated with a given condition

**Cell Array:** A [Matlab data type](https://www.mathworks.com/help/matlab/cell-arrays.html) with indexed data containers called cells, where each cell can contain any type of data.

**User Inputs**

**General Overview**

The user inputs (beapp\_userinputs.m and beapp\_advinputs.m) are where you (“the user”) decide what will happen to your data in BEAPP. You decide:

* Which modules to run on your data (e.g., Do you want to filter your data?)
* Details of what will occur in the modules you turn on (e.g., If you are filtering, do you want a 1 Hz high pass filter? A 200 Hz low pass filter? Both?)

All files must go through the initial formatting step to be run in BEAPP, but most other modules in the pipeline are optional.

* If you turn a module on, you’ll want to make sure the specifications for that module are set to your liking (e.g., if you choose to filter your data, you’ll want to edit the user inputs that determine which frequencies to filter).
* If you turn a module off, you can leave the inputs for that modules set to the default values without impacting your data. (e.g., if you turn the filtering module off, it doesn’t matter how you set your high pass and low pass filter settings – no filtering will occur, regardless of these settings).

Modules in BEAPP are divided into three sections, according to whether data output by a module is:

* Continuous/not yet segmented
* Segmented
* An output metric (e.g. ITPC)

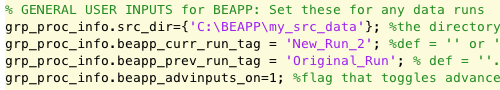
Within a section, the order of the modules will eventually be controlled by the user (functionality not available in BEAPP beta), but modules intended for continuous data cannot be applied to segmented data, and output metrics are calculated from data output by the segmentation module. This is important to keep in mind if you’re reading pre-segmented data into BEAPP, for example, as most preprocessing modules will not be an option (see BEAPP outline figure above).

**Standard User Settings:**

Standard user settings can be found in **/user\_inputs/beapp\_userinputs.m.**

1. First, set the general user inputs (required for every data run)
2. Second, choose the modules to run.
3. Third, choose the details (parameters) of what will occur in the modules you’ve turned on.
4. **General User Inputs (Required for every data run)**

The general user inputs section is required for every data run. It looks like this:

****

In this section, you will need to specify the following information:

* What folder contains the EEG files you plan to process?

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.src\_dir | String format, the source directory for your raw data | **={**‘C:\beapp\_beta\my\_src\_data’}; | Can only have one at a time in Beta version |

* How would you like to label (“tag”) output directories for this data run?

If you’d like directories produced during the current run to have an appended tag, you can specify the tag you’d like to add in grp\_proc\_info.beapp\_curr\_run\_tag. This is recommended – because BEAPP is modular to allow for flexibility in reruns (see section below on “[Rerunning BEAPP Modules](#Rerunning_BEAPP_Modules)” for additional information), it uses directory names to find the data that you’d like to use for each module. Running different data or using different settings in the same set of data directories could cause overwriting of files, or cause you to accidentally use data from a previous run in your current run. This is especially true if you’ve previously run a module that you are no longer running, and it has the same directory tag (or no tag) as your current run. BEAPP will use the data from the previous run in that case, rather than delete your data. Use run tags or clear your directories before you do another run.

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.beapp\_curr\_run\_tag | String that is appended to output directories for this run. | **=**‘2\_sec\_segs’;  =‘eyes\_open’;  Default = ‘’ or ‘NONE’ | During reruns, if this field is left empty, output directories will have a timestamp appended. Enter ‘NONE’ if you would like to mute this feature. |

* If you are re-running data (e.g., you’ve already run data through BEAPP but you want to re-run your data with new filter settings), how was your prior data run labeled (“tagged”)?

Note: For additional information on re-running data, see section below on [“Rerunning BEAPP Modules.”](#Rerunning_BEAPP_Modules)

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.beapp\_prev\_run\_tag | String that was appended to directories that you would like to use as source data. | **=** ‘1\_sec\_windows’; =‘eyes\_open\_only’;  Default = ‘’; | This input is only needed during reruns, to select directories from a previous run that you would like rerun modules to pull data from. |

* Will you be setting any advanced inputs?

Note: For additional information on advanced inputs, see section below on “[Advanced User Settings](#Advanced_User_Settings).”

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.advinputs\_on | Flag on user specification of advanced inputs | = 1; (yes)  =0; (no, use defaults) | Advanced inputs set in beapp\_advinputs.m |

**2. Module Selection**

Module selection is where users determine:

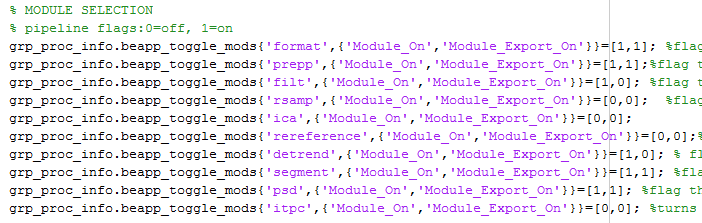
1. Which modules to run

* This is determined by the first value in the vector after the = sign.
  + 1 means “Run this module”
  + 0 means “Don’t run this module”

1. Whether to save the output of each module.

* This is determined by the second value in the vector after the = sign.
  + 1 means “Save output from this module”
  + 0 means “Don’t save output from this module”
* You might choose not to save output from a particular module if that module is an intermediate step in your processing pipeline, if that step can quickly be re-run in the future if needed, and/or if you need to save space on your computer.

The module selection portion of the user inputs looks like this:



In the image above, the user has chosen to run the format, PREP , filtering, detrending, segmentation, and PSD modules. They are saving outputs for format, PREP, segmentation, and PSD, but not filtering or detrending.

For additional information on any of the modules, see the section in the user guide on “Module Parameters” for the module of interest.

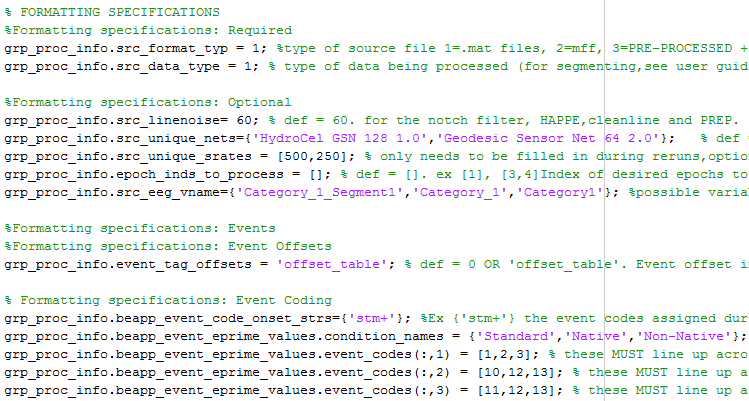
**3. Module Parameters**

Once you’ve chosen which modules to run, look further down in the user inputs to edit the parameters for those modules you’ve turned on. You can ignore the parameters for those modules you’ve turned off.

FORMATTING SPECIFICATIONS

The section on formatting specifications gives BEAPP the information it needs to convert your source EEG data into a format that BEAPP can process. Any new dataset to be run through BEAPP will therefore need to be run through the formatting module.

The section specifying parameters for the formatting module looks like this:



In this section, you will need to specify the following information:

* What is the format of your source files?

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.src\_format\_typ | The format for your raw data. For most users, this will be one of:  1: .Mat exports  2: .MFF files with continuous data  3: .MFF files with segmented data | = 1;  = 2; | Mat exports require an accompanying table (see Running BEAPP with Different Source File Formats) |

* What kind of data will you be processing? (e.g., continuous baseline data, or event-related data)

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.src\_data\_typ | Type of data being processed. For most users, this will be one of:  1 = pure baseline files (pre-extracted)  2 = event related (or baseline with time-locked information + regular event tags)  3 = baseline with blocked conditions (e.g. eyes open, eyes closed blocks) -not available  4 = baseline between event tags or timepoints that is not otherwise marked – not available | = 1;  = 2; | For details and examples, see [Running BEAPP with Differently Structured Data](#Running_BEAPP_With_Differently_Structure). In current version, 3 and 4 are not yet options, but coming soon |

* Which variable(s) contain(s) the EEG data itself? *You only need to specify this if your data is in .mat format.* This is the variable with the (unsegmented) EEG data, in matrix format, as described in the start-up guide.

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.src\_eeg\_vname | List of possible names of the variable with the EEG data | = {'EEG\_Segment1'};  ={‘EEG\_Dat’,‘Cat1\_Seg’} | Only required for .mat files |

* What is the frequency of line noise where your data were collected? *You only need to specify this if you will be running a module that removes line noise (e.g., PREP, Notch filter, or HAPPE).* Typically this is 60 Hz in North and South America, and parts of Asia; it is typically 50 Hz in Europe and other parts of the world. The following website contains information on line noise in other countries: <https://en.wikipedia.org/wiki/Mains_electricity_by_country>
  + If you are running a dataset in which line noise frequency varies by file (e.g., a combined dataset with EEG obtained in the USA and the UK), you will need to specify line noise for each file in mat\_file\_info\_table or mff\_file\_info\_table, as described in the [Start-Up Guide.](#Start_Up_Guide)

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.src\_linenoise | Frequency of line (electrical) noise in your data set, in Hz | **=** 60; (U.S. data, for example)  = 50; (U.K. data, for example) | This is used for notch filtering, HAPPE, and PREP |

* What electrode layouts (or nets) were used to collect your data? *You are only required to specify this if you will be running the ICA module or rereferencing to specific channels.* However, specifying this variable will also increase the speed of runs (especially re-runs). For additional information on managing electrode nets and acquisition layouts in BEAPP, see the section on [BEAPP Net Library](#BEAPP_Net_Library).

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.src\_unique\_nets | List of unique nets present in dataset, EXACTLY as in net library | = {'HydroCel GSN 128 1.0', 'Biosemi 32', 'Biosemi 128'}; | Optional unless running HAPPE/ICA/ adding new net type to library. During reruns, recommended for speed. |

* At what sampling rates were your data collected? *You only need to specify this if you are rerunning a dataset and wish to increase the speed of the rerun.* This should follow the order in .src\_unique\_nets

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.src\_unique\_srates | List of unique sampling rates present in dataset | = [1000,1024,512];  =[250]; | Optional, must align with .src\_unique\_nets |

* Which data recording periods (epochs) would you like to process? *You only need to specify this if your data included multiple recording periods, and you only want to run a subset of those recording periods.* For example, let’s say your continuous EEG included a few minutes of “resting” data, a few minutes of an auditory task, and then a few minutes of a visual task, with each of these tasks in a separate epoch. If you only wanted to run the auditory task, you’d specify that you only wish to run the second epoch.

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.epoch\_inds\_to\_process | Index of epoch to be processed across files. Def = []; (all indexes in all files) | =[1];  =[3,4]; | If baseline data is always in the first epoch, for example |

* What is the offset of events tagged in your dataset? *You only need to specify this if you will be segmenting events; you do not need this for continuous (non-event-related) data.* If you are running a dataset in which offset varies by file, you will need to specify offset for each file in mff\_file\_info\_table, as described in the [start-up guide](#Start_Up_Guide). If the tagging of your events accounts for any offset, you can set this value to zero.
  + Example: Let’s say your EEG has an event tag every time the computer thinks it presents a picture of a face. However, you (as a careful researcher!) have learned that due to a variety of delays in the system wiring, the face in fact appears on the monitor that your experiment’s participant can see 18 milliseconds later. You would therefore set your event tag to 18.

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.event\_tag\_offsets | Event tag offsets in source data, in ms.   * Number if all files have same offset. * If offset differs across files in the dataset, set to ‘offset\_table’ | = 0 ; (no offset in all files)  = 2 ; (offset of 2 ms in all files)  = ‘offset\_table’; (differing offsets) | For information on the format for the offset table, see [Running Event Tagged Data](#Running_Event_Tagged_Data) |

* What is the event code that signifies onset of a stimulus? *You only need to specify this if you will be segmenting events; you do not need this for continuous (non-event-related) data.*

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.beapp\_event\_code\_onset\_strs | List of strings for target event tags to be analyzed | = {‘stm+’};  ={‘faceon’, ‘houseon’}; |  |

* Within a given event, what conditions might exist? *You only need to specify this if you will be segmenting events; you do not need this for continuous (non-event-related) data.*

Note: The impetus for creating the option to manage conditions is that in some cases, the same event code (e.g., stm+) can be used to signify a variety of events. For example, let’s say a participant is having EEG recorded during an oddball task in which they hear three different sounds: A standard syllable, a native syllable, and a nonnative syllable. Onset of all three syllables might be indicated by a stm+ tag, but within that tag there may be event “cell codes” that indicate exactly which of the 3 syllables was delivered. Here, BEAPP provides an opportunity to differentiate among these syllables (based on their “cell codes.”)

Additionally, in some datasets, different cell codes might be used to indicate different event conditions. For example, one site may have indicated Standard, Native, and Non-Native syllables as 1, 2, and 3, respectively. Another site may have indicated these as 10, 11, and 12 respectively. The options here allow BEAPP to handle such an occurrence.

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.beapp\_event\_eprime\_values.condition\_names | Desired condition names for event cell codes (see next line) | ={'Standard', 'Non\_Native', Native}; | Order must match cell codes |
| grp\_proc\_info.beapp\_event\_eprime\_values.event\_codes(:,1)  grp\_proc\_info.beapp\_event\_eprime\_values.event\_codes(:,2)  grp\_proc\_info.beapp\_event\_eprime\_values.event\_codes(:,3) | The cell numbers from E-Prime (or other stimulus presentation program) that correspond to a given condition for event tags. | = [1,2,3];  =[10,12,13];  =[11,12,13]; | See [Running Event Tagged Data](#Running_Event_Tagged_Data) for additional info |

* How should BEAPP recognize time windows or events to be excluded (e.g., events marked in the file as unusable)? This is called “behavioral coding” because in many cases segments of EEG are marked for exclusion based on a behavior that an observer notices during the EEG acquisition (e.g., inattention to a stimulus, blinking, etc). *You only need to specify this if your file contains some information about which events or epochs should be excluded. If you do specify this, trials marked “bad” will be excluded*

|  |  |  |  |
| --- | --- | --- | --- |
| Variables/Inputs | Description | Examples | Notes |
| grp\_proc\_info.behavioral\_coding.events | List of events that include information about behavioral coding. Default = {‘’}; | ={'TRSP'};  ={‘TRSP’,’TRSP2’}; | Optional. |
| grp\_proc\_info.behavioral\_coding.keys | List of keys with behavioral coding in events. | = {'badt'};  ={‘badtf’,’badth’}; | Optional. If filled in, trials with bad values will be rejected |
| grp\_proc\_info.behavioral\_coding.bad\_value | Default = {''};Must be a string | = {‘badt’};  ={‘1’}; | Must be a string |

PREP SPECIFICATIONS

*If the user has turned on the PREP module,* the section on PREP specifications gives BEAPP the information it needs to run the PREP pipeline.(Bigdely-Shamlo et al., 2015). PREP is a very early stage EEG preprocessing pipeline that offers the following:

1. Removal of line noise
2. Detection and interpolation of bad channels
3. Robust average referencing

The PREP pipeline is standalone software that has been integrated into BEAPP. The majority of PREP settings are determined by its intrinsic defaults, although the user does have the option to change the line noise frequency as described in the format module above. Otherwise, the only determination the user needs to make is whether they would like BEAPP to create an Excel output of PREP’s findings. Therefore, the section specifying parameters for the PREP module looks like this:



In this section, the user will need to specify only the following information:

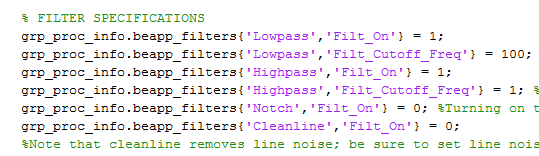
* Do you want BEAPP to save out a report table with PREP summary information about the files used?
  + 1 means “Save summary output from PREP”
  + 0 means “Don’t save summary output from PREP”

FILTER SPECIFICATIONS

*If the user has turned on the filtering module,* the section on filter specifications determines filter settings to be run on the dataset. BEAPP offers three types of filtering, and offers Cleanline for line noise removal:

1. Low pass filtering (essentially allowing only oscillations slower than a given frequency to be included)
2. High pass filtering (essentially allowing only oscillations faster than a given frequency to be included)
3. Notch filtering (for line noise removal)
4. [Cleanline](https://bitbucket.org/tmullen/cleanline) (line noise removal)

The section specifying parameters for the filter module looks like this:



In this section, for each filter type (notch, low pass, high pass) the user will need to specify only the following information:

* Should this filter type be turned on?
  + 1 means “Turn on this filter type”
  + 0 means “Turn off this filter type
* What is the filter’s cutoff frequency? (Of note, this is only specified for low pass and high pass filters. Notch filtering or Cleanline, if turned on, automatically occur at the line noise frequency).

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.beapp\_filters{'Lowpass','Filt\_On'} | Flags lowpass filter on and off | =0; (off)  =1; (on) |  |
| grp\_proc\_info.beapp\_filters  {'Lowpass','Filt\_Cutoff\_Freq'} | Cutoff frequency for lowpass filter | =100; |  |
| grp\_proc\_info.beapp\_filters{'Highpass','Filt\_On'} | Flags highpass filter on and off | =0; (off)  =1; (on) |  |
| grp\_proc\_info.beapp\_filters  {'Highpass','Filt\_Cutoff\_Freq'} | Cutoff frequency for highpass filter | =1; |  |
| grp\_proc\_info.beapp\_filters{'Notch','Filt\_On'} | Flags notch filter on and off | =0; (off)  =1; (on) | When on, BEAPP notch filters at line noise frequency |
| grp\_proc\_info.beapp\_filters{'Cleanline','Filt\_On'} | Flags Cleanline on and off | =0; (off)  =1; (on) | When on, BEAPP applies at line noise frequency |

Note: Keep in mind that these settings only matter if the filtering module is turned on. If the filtering module is turned off in the “Module Selection” section, no filtering will occur (even if individual filters are set to “1”).

Note: Many users will likely choose to low pass filter at the Nyquist frequency (i.e., just under half of the sampling rate). If downsampling, many users will likely choose to low pass filter at the Nyquist frequency of the target sampling rate. For example, if the sampling rate is 250 Hz (or if the sampling rate is 500 Hz but the user will resample to 250 Hz), a user might low pass filter at 100 Hz.

RESAMPLING SPECIFICATIONS

*If the user has turned on the resampling module,* the section on resampling specifications determines how data will be resampled.

The section specifying parameters for the resampling module looks like this:



In this section, the user will need to specify only the following information:

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.beapp\_rsamp\_srate | Desired sampling rate (Hz) after resampling | =250; |  |

Note: Many users will likely choose to resample if they have EEGs in a dataset that are collected at a variety of sampling rates. In many such cases, users will downsample all data to the lowest common sampling rate. For example, if a user has EEG data sampled at 1024 Hz, 100 Hz, 500 Hz, and 250 Hz, they might choose to downsample all data to 250 Hz.

A user might also downsample to improve the quality of ICA decomposition. Some ICA paradigms work best at lower sampling rates (e.g., 250 Hz).

ICA/HAPPE SPECIFICATIONS

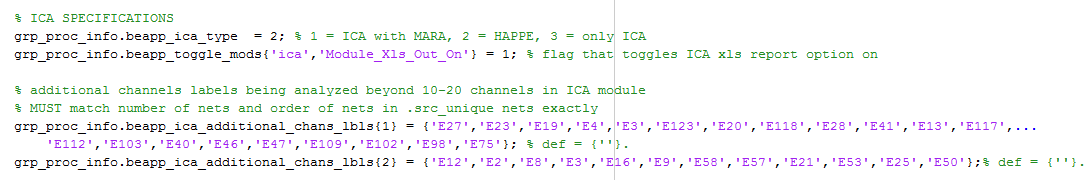
Users have the option to run ICA, ICA+MARA, or the Harvard Automated Preprocessing Pipeline for EEG (HAPPE), a specific EEG preprocessing pipeline targeted towards artifact removal for infant EEG data.

HAPPE includes the following steps:

1. Identify channels in the 10-20 electrode system (for EEG recorded from higher-density systems), and any other electrodes to be included.
   1. Note: This step takes place during the formatting module, described above. For additional information on selecting these channels, and any additional channels to be included in the HAPPE analysis, see the section below.
2. 1 Hz high pass filtering, and 250 Hz low pass filtering
   1. Note: If a user wishes to run HAPPE, the filtering module should be turned on with these settings, and with the notch filter off.
3. Line noise removal using the *cleanline* function
   1. Note: Line noise frequency is set in the formatting module, described above. However, unless a user runs PREP or a notch filter (neither of which is recommended if a user will be running HAPPE), line noise removal does not take place until the HAPPE module runs.
4. Crude bad channel detection
5. Wavelet cleaning
6. ICA with MARA
7. Interpolation of bad channels
8. Reference to average
9. Removal of bad segments of data

**Note: Instructions for running the ICA module in BEAPP are included on this page and the next. Additional information required to run HAPPE is included on the page following this section.**

The section specifying parameters for the ICA module looks like this:



Running the ICA/ICA+MARA/HAPPE module in BEAPP:

* Which ICA process would you like to run?

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.beapp\_ica\_type | 1 = ICA with MARA  2 = HAPPE  3 = only ICA |  |  |

If you choose to run ICA+ MARA, the other steps of HAPPE described above (i.e., line noise removal, crude bad channel detection, wavelet cleaning, interpolation of bad channels, average reference) are not included.

* Do you want BEAPP to save out a report table with summary information about the files used?

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.beapp\_toggle\_mods{'ica','Module\_Xls\_Out\_On'} | Toggles HAPPE summary output table on and off | =0; (off)  =1; (on) | When on, BEAPP saves out a report table with summary information about the files used. |

* In addition to the 10-20 channels, would you like ICA to include any additional channels in its analyses?

Note: The ICA module will automatically use any electrodes listed as equivalent to 10-20 electrodes for a given net in the net library (as MARA uses 10-20 locations for component evaluation). Users have the option to add additional electrodes of interest for each net using grp\_proc\_info.happe\_additional\_chans\_lbls. Channels should be listed using the EEGLAB labels (e.g ‘E41’). Each net’s cell number in grp\_proc\_info.ica\_additional\_chans\_lbls (e.g. grp\_proc\_info.ica\_additional\_chans\_lbls{1}) should correspond to the net’s position in grp\_proc\_info.src\_unique\_nets. The number of channels listed for each net can be different. For now this is applied to ICA without MARA as well.

Choosing how many additional channels to run in the ICA module is somewhat subjective, and users can run the module with visualizations on to check how well the ICA decomposition is working when there are different numbers of channels as input. However, generally, the number of channels that can be added will be limited by the input file’s length (time). It is recommended that the number of samples in your EEG recording be equal to 20\* (the number of channels)2.

- For example, an EEG acquired with a 128-channel net and sampling rate of 500 Hz (500 samples/ second) would need at least 327,680 samples (1282 \* 20 samples), that is, 655.36 seconds of recording (327,680 samples at 500 Hz) reliably decompose all channels with ICA (see HAPPE for further discussion).

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.ica\_additional\_chans\_lbls{1}  grp\_proc\_info.ica\_additional\_chans\_lbls{2}  etc. | Labels associated with additional (non-10-20 configuration) channels in ICA. Each row applies to the corresponding net from .src\_unique\_nets; | ={‘E119’}  ={‘E36’ ‘E42’ ‘E8’}; |  |

Example: Let’s say a user is running ICA for 2 nets, using different electrodes for each net. In formatting specifications, they specified nets as follows:



If the user wishes to include additional channels (not just the 10-20 channels) in the ICA decomposition, they could therefore specify additional channels as follows:



Note that the first set specifies channels to be included for EEG data from a HydroCel GSN 128 1.0 net, and the second set specifies channels to be included for EEG data from a Geodesic Sensor Net 64 2.0. The order of specification is important here.

Notes regarding channel inclusion:

* ICA is the only module in BEAPP that automatically restructures data to include only the 10-20 channels (along with any additional channels that the user explicitly instructs BEAPP to include).
* All other modules in BEAPP will include all input channels (i.e., all 64 channels in a 64-channel net, and all 128 channels in a 128-channel net).
* Outside of HAPPE, the only exception to the rule that BEAPP will include all channels is if the user specifies (in advanced user inputs) that they would like bad channels removed. In this case, any channels that PREP (or another module) identifies as bad will be replaced with NaN. Otherwise, BEAPP defaults to interpolating channels that PREP (or another module) identifies as bad, but then keeps these interpolated channels for downstream analysis.

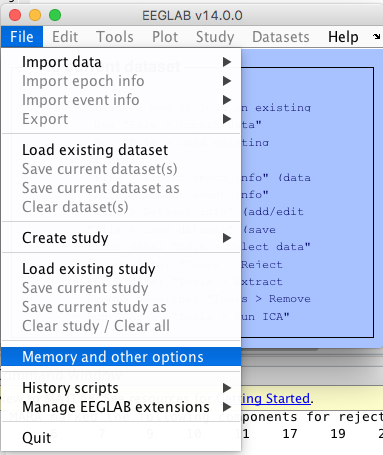
HAPPE/ICA-MARA Advanced Inputs:

* Advanced users may choose to flag on the visualizations provided by MARA (which will then require manual user selection of components for each file), in beapp\_advinputs.m. as follows

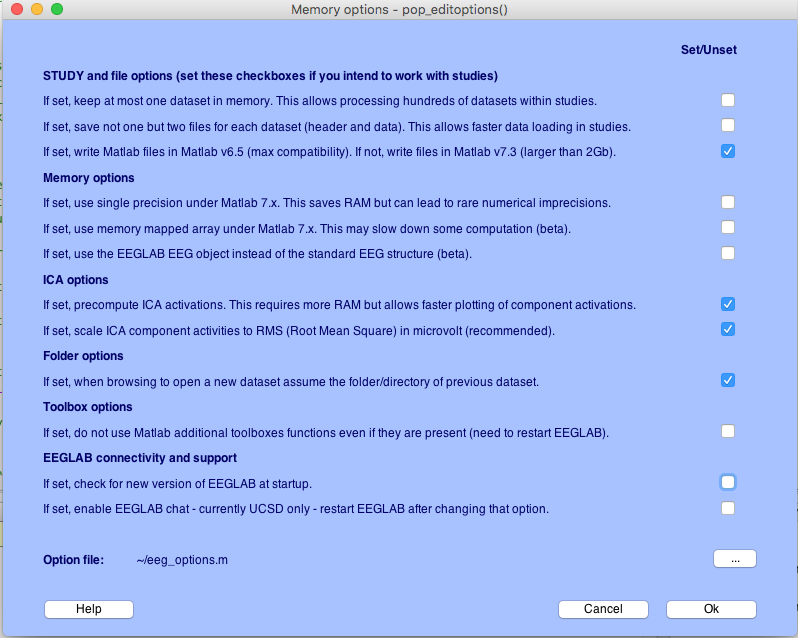
|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.happe\_plotting\_on | Turns HAPPE visualizations on and off | = 0; (off)  =1; (on) |  |

**Note: Before running the ICA module for the first time, follow the instructions below. The page thereafter includes information on HAPPE settings in BEAPP’s user inputs.**

**Before running HAPPE for the first time,** users will need to configure EEGLAB memory settings. After opening the EEGLAB GUI (type eeglab into matlab command window to open the GUI), users should navigate to File-> Memory and other options.



Settings for this panel should match the image below and click ok before quitting EEGLAB:



Once you’ve set up EEGLAB for HAPPE, you’re ready to return to your user inputs.

Additional note: Several of the BEAPP modules outside of HAPPE will impact how HAPPE is run. The creators of HAPPE have the following recommendations:

**Formatting module:**

Users running HAPPE or ICA with MARA in BEAPP will need to have included accurate 10-20 electrode mappings when adding the appropriate nets to the net library (see the section below titled [BEAPP Net Library](#BEAPP_Net_Library)). Additionally, if HAPPE is being run on a dataset with more than one EEG acquisition layout or net, you will need to set the following inputs accordingly:

|  |  |  |
| --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** |
| grp\_proc\_info.src\_unique\_nets | List of unique nets present in dataset, EXACTLY as in net library | = {'HydroCel GSN 128 1.0', 'Biosemi 32', 'Biosemi 128'}; |

The line noise removal portion of HAPPE will also refer to the line noise frequency specified in the input module.

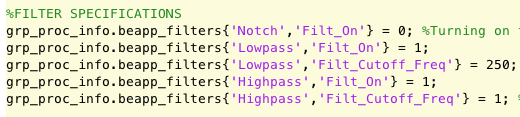
|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.src\_linenoise | Frequency of line (electrical) noise in your data set, in Hz | **=** 60; (U.S. data, for example)  = 50; (U.K. data, for example) | This is used for notch filtering, cleanline, HAPPE, and PREP |

**PREP module:**

Because PREP interpolates bad channels, and because previously interpolated channels alter the integrity of ICA-MARA, it is recommended that the PREP module be turned off if a user will be running HAPPE.

**Filtering module:**

HAPPE recommends the following filtering settings:



**Resampling module:**

HAPPE recommends the following resampling settings:



**Segmenting module:**

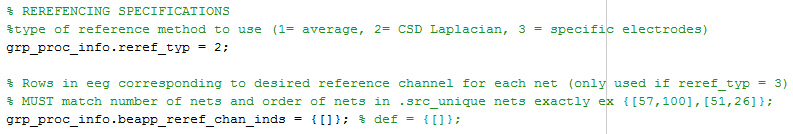
HAPPE recommends the following artifact threshold:



REREFERENCING SPECIFICATIONS

*If the user has turned on the rereferencing module,* the section on rereferencing specifications determines the type of rereferencing that will occur.

The section specifying parameters for the detrending module looks like this:



In this section, the user will need to specify only the following information:

* What type of rereferencing would you like applied to your data?

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.reref\_typ | Detrending method applied in module:  1 = average  2= CSD Laplacian  3 = to electrode(s) | =1;  =2;  =3; | CSD uses the CSD Toolbox |

* Users rereferencing to a specific electrode or electrodes must specify which electrode(s) they would like to use in each net. These must line up exactly with grp\_proc\_info.src\_unique\_nets, specified in the formatting module

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.beapp\_reref\_chan\_inds | Rows in eeg corresponding to desired reference channel for each net | ={[]};  = {[57,100],[51,26]}; | Order must match .src\_unique\_nets exactly |

DETRENDING SPECIFICATIONS

*If the user has turned on the detrending module,* the section on detrending specifications determines the type of detrending that will occur.

The section specifying parameters for the detrending module looks like this:



In this section, the user will need to specify only the following information:

* What type of detrending would you like applied to your data?

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.dtrend\_typ | Detrending method applied in module:  1 = mean  2= linear  3 = Kalman | =1;  =2;  =3; |  |

Detrending Advanced Inputs:

Several advanced inputs specific to the detrending module are included in beapp\_advinputs.m. These include the following inputs, specific to the Kalman filter.

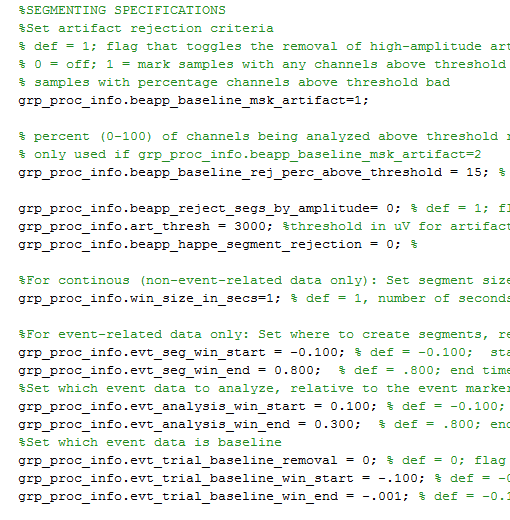
|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.kalman\_b | B value for the Kalman filter detrend option. | =.9999; | Value chosen will depend on sampling rate. Use caution; lower b-values will improve fitting (and improve number of useable segments), but may excessively attenuate low and even mid-frequency oscillations. |
| grp\_proc\_info.q\_init | Determines smoothing in the Kalman filter | =1; | Default is 1 |

Note on Kalman filtering: This will likely be used only in specific circumstances (e.g., to attenuate the impact of TMS artifact or certain epileptiform activity). Lower b-values will improve fitting, and therefore decrease the impact of high-amplitude artifact. For users excluding segments using amplitude cutoffs, lower b-values may therefore increase the number of useable segments. However, users should be cautious about the potential impact of low b-values on the power spectrum; low b-values may excessively attenuate low (e.g., delta, theta) and even mid (e.g., alpha, beta) frequency oscillations. The extent of this effect may also depend on the sampling rate. Therefore, if users plan to use a Kalman filter and then run the PSD (power spectral density) module, we encourage them to plot power spectra using several different b-values, to determine potential impact of the chosen value on the power spectrum.

SEGMENTING SPECIFICATIONS

*If the user has turned on the segmentation module,* the section on segmentation specifications determines the type of segmentation that will occur.

The section specifying parameters for the detrending module looks like this:



In this section, the user will need to specify the following information:

* Will high amplitude artifact be removed prior to segmentation (in baseline/resting data)?

Note: If this is turned on, BEAPP will create a mask of all unusable data prior to segmentation. To do so, BEAPP will scan the data in all channels for any data point that is above threshold. Upon identifying these suprathreshold data points, BEAPP will determine the nearest zero-crossing before and after that data point, for that channel. Above-threshold segments are then defined as beginning and ending at the nearest zero-crossings, rather than only including the narrower windows of time where data is suprathreshold. If .beapp\_msk\_artifact =1, at each data point, if an above-threshold segment is marked in any channel, that data point is determined to be unusable. If .beapp\_baseline\_msk\_artifact =2, at each data point, if above-threshold segments are marked in more than the user-set percent of channels in .beapp\_baseline\_rej\_perc\_above\_threshold, that data point is determined to be unusuable. Segments will be created only from the usable data, as specified below.

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.beapp\_baseline\_msk\_artifact | Flag on BEAPP amplitude-based artifact removal software during segment. 0= off; 1 = reject if 1 or more bad channels, 2 = reject if more than a % of channels are bad | =0; (no)  = 1; (yes, reject samples with any bad channels)  = 2; (yes, reject samples with more than a % bad channels) | BEAPP artifact detection occurs during the segmentation module |
| Grp\_proc\_info.beapp\_baseline\_rej\_perc\_above\_threshold | Percentage of bad channels required to mark sample bad | =15; (15%)  =1; (1%) | Only used if above = 2 |

* What is the amplitude threshold for artifact removal?

The default value is set to 100. However, users should note that certain types of data (e.g., infant data, or EEG recorded after craniotomy) may require higher amplitude thresholds (e.g., 150). Other types of data (e.g., data that has been run through HAPPE, or data that has undergone a Laplacian transform) may require changes to amplitude thresholds. For HAPPE, an amplitude threshold of 40 or 50 is typically recommended.

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.art\_thresh: | Threshold (in mV) for BEAPP artifact removal software. Default = 100; | =100;  =150; | Will need to be scaled appropriately if using CSDLP (e.g. 3000) |

* Will high amplitude artifact be removed after segmentation?

Note: If this is turned on, BEAPP will first create segments as specified below. It will then remove any segments in which amplitude is above the specified threshold in any channel.

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info. beapp\_reject\_segs\_by\_amplitude | Threshold (in mV) for BEAPP artifact removal software. Default = 100; | =100;  =150; | Will need to be scaled appropriately if using CSDLP (e.g. 3000) |

* Do you want to use the HAPPE segment rejection code? This rejects based on amplitude after segmentation, and the joint probability of samples (EEGLAB jointprob function)

Note: Some users might choose not to use this code if they wish to keep all segments, or reject segments using another set of parameters (e.g., amplitude-based rejection criteria from the segmenting module only).

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.beapp\_happe\_segment\_rejection | Toggles HAPPE segment rejection code on and off | =0; (off)  =1; (on) | In BEAPP, the HAPPE segment rejection occurs in the segmentation module |

Segment Size Parameters:

* What will be the segment size? *This only needs to be specified for continuous (non-event-related data). For event-related data, segment size will be determined by segment start and end times as they relate to the event marker of interest*

Note: This is often set to either 1 or 2 seconds.

* + For infant data, which is often highly contaminated by artifact, 1-second segments may be more feasible.
  + If the user will be running the PSD module with a multitaper, the minimum meaningful number of tapers (3) requires segment length of at least 2 seconds in order to get a frequency resolution of 1 Hz. For most users looking at low (delta, theta) or mid (alpha, beta) frequencies, segments of at least 2 seconds will therefore be necessary for a multitaper. However, for users looking at higher (gamma) frequencies, where low frequency resolution is less of a concern, a 1-second window may still be feasible with 3 tapers.
  + Users interested in looking at low frequencies (e.g., delta and below) may wish to use segments longer than 2 seconds.

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.win\_size\_in\_secs | Window size (in seconds) for segment creation and windowed analyses | = 1;  = 2; | Used in PSD, etc. |

* Where will segments start and end in relation to the event marker of interest? *This only needs to be specified for event-related data.*

Note: This specifies how the data will be segmented, which may be different from how it will be analyzed. A segment should include all the data to be analyzed, but the analysis may include only a portion of the segment. For example, if a user wishes for the analysis window (e.g., 100 to 800ms post-stimulus) to be compared to a baseline window (e.g.,-200 to -100ms), then the segment should start at -200ms (or earlier), and the segment should end at 800ms (or later). These inputs determine the size of segments saved after this module

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.evt\_seg\_win\_start | Segment start (in seconds) for event-related segments, relative to event of interest | = - 0.2; |  |
| grp\_proc\_info.evt\_seg\_win\_end | Segment end (in seconds) for event-related segments, relative to event of interest | = 0.8 |  |

* Where will the analysis window start and end in relation to the event marker of interest? *This only needs to be specified for event-related data.*

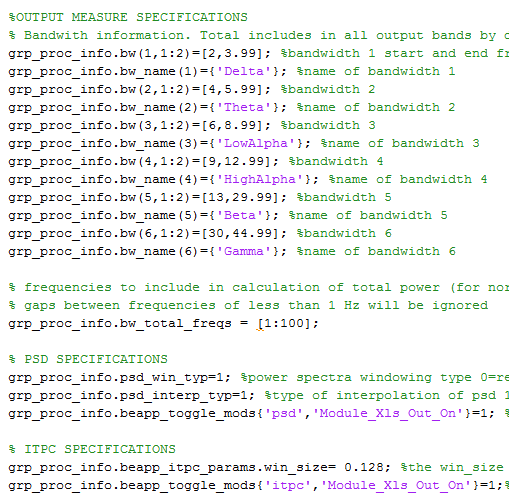
|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.evt\_analysis\_win\_start | Analysis window start (in seconds) for event-related segments, relative to event of interest | = 0.1; | Must be greater than (or equal to) segment start time specified above |
| grp\_proc\_info.evt\_analysis\_win\_end | Analysis window end (in seconds) for event-related segments, relative to event of interest | = 0.8 | Must be less than (or equal to) segment end time specified above |

* Will data be baseline corrected? If so, where will the baseline window start and end in relation to the event marker of interest? *This only needs to be specified for event-related data, and only if the output calculations will involve baseline correction.*

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.evt\_trial\_baseline\_removal | Flag on baseline removal for individual segments | =0; (no)  =1; (yes) |  |
| grp\_proc\_info.evt\_trial\_baseline\_win\_start | Baseline window start (in seconds) for event-related segments, relative to event of interest | = -0.2 | Must be greater than (or equal to) segment start time specified above |
| grp\_proc\_info.evt\_trial\_baseline\_win\_end | Baseline window end (in seconds) for event-related segments, relative to event of interest | = -0.1 | Must be less than (or equal to) segment end time specified above |

OUTPUT MEASURE SPECIFICATIONS

*If the user has turned on any output modules (e.g., PSD or ITPC),* the section on output module specifications determines the parameters of these output measures. The section specifying parameters for the output measures looks like this:



In this section, the user will need to specify the following information:

* What are the bandwidths of interest? *This is only necessary if the user wants csv reports for the PSD or ITPC modules (or other output modules in the future).* The user can specify as many (or as few) frequency bands of interest as they wish. The user is also asked to specify which frequencies they would like to include in measures of total power (used to normalize power in reports)

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.bw(1,1:2)  grp\_proc\_info.bw\_name(1)  etc. | Start and end frequencies and names for each bandwith you would like to analyze- can have as many or few as you add. | grp\_proc\_info.bw(1,1:2)=[2,4];  grp\_proc\_info.bw\_name(1)={'Delta'};  grp\_proc\_info.bw(2,1:2)=[4,6];  grp\_proc\_info.bw\_name(2)={'Theta'};  grp\_proc\_info.bw(3,1:2)=[6,9];  grp\_proc\_info.bw\_name(3)={'LowAlpha'};  grp\_proc\_info.bw(4,1:2)=[9,13];  grp\_proc\_info.bw\_name(4)={'HighAlpha'};  grp\_proc\_info.bw(5,1:2)=[13,30];  grp\_proc\_info.bw\_name(5)={'Beta'};  grp\_proc\_info.bw(6,1:2)=[30,50];  grp\_proc\_info.bw\_name(6)={'Gamma'}; | Total (all frequencies between the start of the lowest bandwith and the end of the highest) is included automatically |
| grp\_proc\_info.bw\_total\_freqs | Frequencies to include in total power | = [1:100]; (default)  =[1.5:57,63:110]; | Gaps of less than 1 Hz are ignored |

SPECIFIC OUTPUT MODULES:

If the user wishes to obtain data on the POWER SPECTRUM, the following additional specifications are necessary:

* What type of windowing should be applied to calculate the power spectrum?

Note: Most signal processors recommend a multitaper if possible. This smooths the power spectrum, and mitigates edge effects. However, as described in the section above on [choosing segment length,](#Choosing_Seg_Length) some users may find that their segment lengths are too short, and frequency resolution needs too high, to allow for a multitaper to be feasible.

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.psd\_win\_typ | Window type for power spectra:  0 = Rectangular  1 = Hanning  2 = multitaper | =1;  =2; | 2+ second windows length recommended for multitaper, to allow frequency resolution of 1 Hz. |

* Do you want BEAPP to interpolate the power spectrum?

Most users will not choose any interpolation. Note that interpolation does not change the frequency resolution of the PSD; it simply interpolates between values output by the PSD. In some cases, however, users may wish to interpolate the frequency axis if they are concerned about the effects of “edge” values in binned power calculations. For example, if power values are output at 2, 4, and 6 Hz in the example above, interpolation might help a user better differentiate between power in the delta vs. theta vs. low alpha bands.

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.psd\_interp\_typ | Type of interpolation for the psd:  1 = None  2 = linear  3 = nearest neighbor  4 = piecewise cubic spline | =1;  =2;  =3;  =4; | This interpolates the power spectrum between output frequencies, but does not alter frequency resolution of the power spectrum. |

* Do you want BEAPP to save out a report table with PSD output summary information?

This table would include power values in each frequency band, in each channel, for each file.

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.beapp\_toggle\_mods{'psd','Module\_Xls\_Out\_On'} | Toggles PSD summary output table on and off | =0; (off)  =1; (on) | When on, BEAPP saves out a report table with PSD summary information |

If the user wishes to obtain data on INTER-TRIAL PHASE COHERENCE (ITPC), the following additional specifications are necessary:

* What size sub-window should be applied to calculate ITPC?

When calculating ITPC, the analysis window is typically divided into multiple overlapping sub-windows, each of the size specified here. Thus ITPC is not output as a single value, but instead as a vector of values specifying the ITPC for the midpoint of each of these sub-windows. This allows the user to assess how ITPC changes over a window of time in relation to a repeated event.

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.beapp\_itpc\_params.win\_size | Window size in seconds used to calculate ITPC | =0.256; |  |

* Do you want BEAPP to save out a report table with ITPC output summary information?

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.beapp\_toggle\_mods{'itpc','Module\_Xls\_Out\_On'} | Toggles ITPC summary output table on and off | =0; (off)  =1; (on) | When on, BEAPP saves out a report table with ITPC summary information |

**Advanced User Settings:**

Advanced user inputs can be found in \user\_inputs\beapp\_advinputs.m. These will only be applied if grp\_proc\_info.adv\_inputs =1 in the standard user inputs. Otherwise, the defaults will be used for these values. Advanced users not wanting to use more than one or two advanced inputs can also copy these to the standard user inputs if they want to set certain values without turning the advanced inputs on.

Advanced General Variables/Inputs:

* Mute directory warnings?

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.beapp\_dir\_warn\_off | Mutes BEAPP directory warnings that require user input. | =0; (default)  =1; | BEAPP will not alert the user when there is a risk of data being overridden if this is on |

* Only use a subselection of files previously run during a rerun?

Note: by default, this uses the rerun\_fselect\_table.mat in user\_inputs to determine which files to rerun. See [Rerunning BEAPP Modules](#Rerunning_BEAPP_Modules) for additional information

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.use\_rerun\_table | Use a file selection table for a subset of files | =0; (default)  =1; (on) | Optional input only used during reruns |

* Replace bad channels with NaN instead of interpolating? Users may choose to do this if they would like to apply analyses without interpolated channels

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.beapp\_rmv\_bad\_chan\_on | Replace bad channels with NaN instead of interpolating. | =0; (no, default)  = 1; (yes) | Applied in PREP or HAPPE |

Advanced Inputs by Module:

Formatting Specifications:

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.mff\_seg\_throw\_out\_bad\_segments | Throw out segments previously marked bad (during hand editing, for example) during import? | =1; (yes, default)  = 0; (no) | Only used during format for pre-segmented MFFs |

Filter Specifications:

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.src\_buff\_start\_nsec | Number of seconds for buffer at start of EEG to be excluded after filtering and artifact removal | =2;  =0; | Should only be 0 if no filtering applied. Otherwise need to account for filter rolloff. |
| grp\_proc\_info.src\_buff\_end\_nsec | Number of seconds for buffer at end of EEG to be excluded after filtering and artifact removal | =2;  =0; | Should only be 0 if no filtering applied. Otherwise need to account for filter rolloff. |

ICA/HAPPE/MARA Specifications:

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.happe\_plotting\_on | Turns MARA visualizations on and off | =1;  =0; | Allows visualization of each processing step in HAPPE. If this is on, each file will require user input to confirm the rejection of ICA components during HAPPE |

Rereference Specifications:

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.beapp\_csdlp\_interp\_flex | Spline flexibility for CSD toolbox. m =2-10, default =4 | =4; |  |

Detrending Specifications:

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.kalman\_b | B value for the Kalman filter detrend option. | =.9999; | Value chosen will depend on sampling rate. Use caution; lower b-values will improve fitting (and improve number of useable segments), but may excessively attenuate low and even mid-frequency oscillations. |
| grp\_proc\_info.q\_init | Determines smoothing in the Kalman filter | =1; | Default is 1 |

Segmenting Specifications;

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.segment\_linear\_detrend | Detrend each channel in each segment after segmentation | =0; (no)  =1; (linear)  =2; (mean) |  |
| grp\_proc\_info.beapp\_erp\_maf\_on | Turns the moving average filter on for event related data | =1; (yes)  =0; (no) |  |
| grp\_proc\_info.beapp\_erp\_maf\_order | Order of the moving average filter | =30; | Default is 30 |

Advanced User Inputs for BEAPP Outputs (PSD, ITPC, etc.):

PSD Specifications:

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.psd\_pmtm\_l | Number of tapers to use if using the multitaper window | = 3; | Should be integer ≥ 3 |
| grp\_proc\_info.beapp\_xlsout\_av\_on | Report mean power in .csv exports for the PSD module | =1; (on)  =0; (off) |  |
| grp\_proc\_info.beapp\_xlsout\_sd\_on | Report standard deviation in .csv exports for the PSD module | =1; (on)  =0; (off) |  |
| grp\_proc\_info.beapp\_xlsout\_raw\_on | Report raw power in .csv exports for the PSD module | =1; (on)  =0; (off) |  |
| grp\_proc\_info.beapp\_xlsout\_norm\_on | Report normalized power in .csv exports for the PSD module, with each frequency band normalized to total power | =1; (on)  =0; (off) |  |
| grp\_proc\_info.beapp\_xlsout\_log\_on | Report natural log power in .csv exports for the PSD module | =1; (on)  =0; (off) |  |
| grp\_proc\_info.beapp\_xlsout\_log10\_on | Report log10 power in .csv exports for the PSD module | =1; (on)  =0; (off) |  |

ITPC Specifications:

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.beapp\_itpc\_xlsout\_mx\_on | Report mean ITPC in Excel exports for the ITPC module | =1; (on)  =0; (off) |  |
| grp\_proc\_info.beapp\_itpc\_xlsout\_av\_on | Report standard deviation of ITPC in Excel exports for the ITPC module | =1; (on)  =0; (off) |  |

**Rerunning BEAPP modules**

If you have run BEAPP on a dataset, and would like to rerun some section of the pipeline with new parameters or files, without rerunning all of the modules upstream, you can run BEAPP as normal with only the desired modules selected.

BEAPP automatically appends new directories created during reruns with a timestamp. You can change the tag added to directories from a timestamp using the grp\_proc\_info.beapp\_curr\_run\_tag input, or mute the appended tag (choosing to overwrite the files for the modules you’re rerunning) by setting grp\_proc\_info.beapp\_curr\_run\_tag = ‘NONE’. If you’d like to rerun modules from directories that have a tag appended to them (directories from runs where grp\_proc\_info.beapp\_curr\_run\_tag was not empty, or where timestamps were appended), you can specify this using grp\_proc\_info.beapp\_prev\_run\_tag.

During reruns, it’s recommended that you specify the list of net types present in your dataset (exact names as they appear in your [net library](#BEAPP_Net_Library)) in grp\_proc\_info.src\_unique\_nets, and the list of sampling rates present in the dataset in grp\_proc\_info.unique\_srates. While specifying these values is not required for BEAPP to run, it will make your rerun much faster, especially for large datasets. It’s fine to include this information during initial runs, but it will not impact run speed.

Finally, if you would only like to rerun the selected modules for a subset of the files in the most recent source directory, you may choose to set the advanced input grp\_proc\_info.beapp\_use\_rerun\_table = 1. By default, this will use the files listed in the rerun\_fselect\_table.mat in user\_inputs (and optionally, net types, sampling rates, and linenoise frequencies). For information on generating this table, see [Generating a mat\_file\_info\_table](#Gen_mat_file_info_table)”).

**Saving/Reusing User Input Templates or File Information Tables**

**Saving Templates**

Some users may choose to save their user settings for a dataset to allow them to switch between setting configurations quickly without having to re-enter information. Users wishing to do this should enter their settings in beapp\_userinputs.m and beapp\_advinputs.m as usual, and then save each file with a new name using Save As in Matlab (found in the Save menu in the Editor panel).

**Saving File Information Tables**

Some users may choose to save the mat\_file\_info\_tables or file offset tables used for different datasets as well. Users wishing to do this should create the tables as normal, and rename the .mat file with the desired table name (as described in “[Generating a mat\_file\_info\_table](#Gen_mat_file_info_table)”). **Make sure not to change the Matlab variable name.**

**Using Templates and Saved Tables**

If you’d like to change between multiple user input or advanced input templates, or change which table is used as the mat\_file\_info\_table or the mff\_file\_info\_table, you may specify the file paths for your templates in /user\_inputs/beapp\_set\_input\_file\_locations.m. Any of these left blank will use the standard BEAPP files/tables – **make sure that if you don’t want to use a different input template or table, these are set blank before your run.**

|  |  |  |
| --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** |
| grp\_proc\_info.beapp\_alt\_user\_input\_location | File path for alternative BEAPP user inputs template | {‘C:\example\_dir\user\_inputs\dataset2\_settings.m’}; |
| grp\_proc\_info.beapp\_alt\_adv\_user\_input\_location | File path for alternative BEAPP advanced inputs template | {‘C:\example\_dir\user\_inputs\dataset2\_adv\_settings.m’}; |
| grp\_proc\_info.beapp\_alt\_mat\_file\_info\_table\_location | File path for alternative BEAPP mat\_file\_info\_table | {‘C:\example\_dir\user\_inputs\dataset2\_mat\_file\_info\_table.m’}; |
| grp\_proc\_info.beapp\_alt\_mff\_file\_info\_table\_location | File path for alternative BEAPP mff\_file\_info\_table | {‘C:\example\_dir\user\_inputs\dataset2\_mff\_file\_info\_table.m’}; |
| grp\_proc\_info.beapp\_alt\_rerun\_fselect\_table\_location | File path for alternative BEAPP rerun\_fselect\_table | {‘C:\example\_dir\user\_inputs\dataset2\_rerun\_fselect\_table.m’}; |

**Running BEAPP with Different Source File Formats**

All source files for a given run should be placed in the source directory listed in the user inputs.

**Unsegmented MFF (EGI) Files (Baseline or Event Related, 1 or More NetStation Epochs)**

To run unsegmented MFF files you’ll need to enter the appropriate source directory and user settings, and set grp\_proc\_info.src\_format\_typ = 2. MFF files with multiple NetStation epochs can be read by BEAPP using this setting. If your files have different event tag offsets, you’ll need to indicate individual file names and file offsets in the mff\_file\_info\_table in the user inputs, and change grp\_proc\_info.event\_tag\_offsets = ‘offset\_table.’ You can also enter the exact names of possible net types into grp\_proc\_info.src\_unique\_nets for speed, although it’s optional if you’re not running HAPPE.

**Segmented MFF (EGI) Files (Baseline or Event Related)**

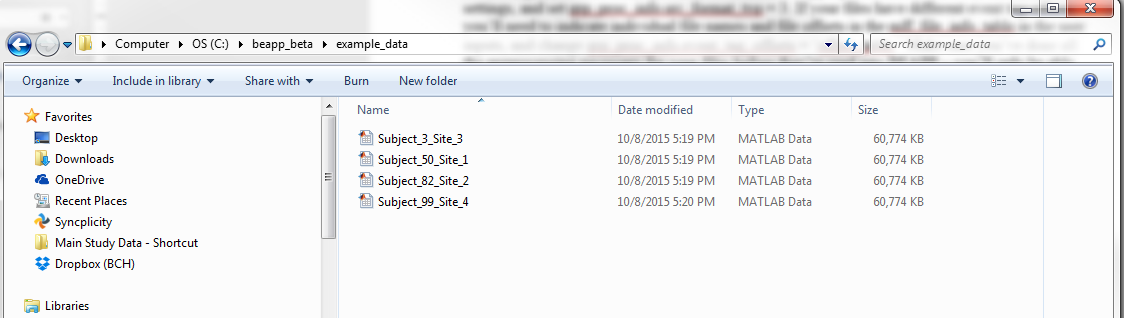
To run segmented MFF files you’ll need to enter the appropriate source directory and user settings, and set grp\_proc\_info.src\_format\_typ = 3. If your files have different event tag offsets, you’ll need to indicate individual file names and file offsets in the mff\_file\_info\_table in the user inputs, and change grp\_proc\_info.event\_tag\_offsets = ‘offset\_table.’ Make sure you’ve done all the preprocessing necessary for your files before they’re read into BEAPP – you’ll only be able to use BEAPP output metrics, since the files are already segmented.

**.Mat exports (from EGI, Biosemi, ANT, etc.)**

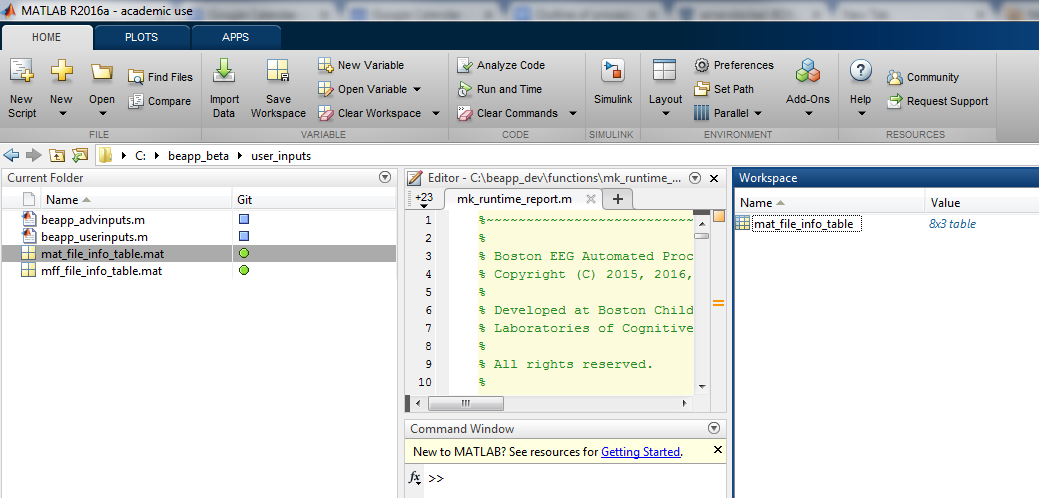
BEAPP has the ability to run Matlab file exports from a variety of acquisition setups. Each .mat file must contain a variable with the (unsegmented) EEG data. The name of this variable can be whatever you typically use in your exports, but should be consistent across files (even if your dataset contains files from mixed acquisition setups). You should specify the name(s) of this variable in grp\_proc\_info.src\_eeg\_vname. In addition, for all files you would like to run, you must add a line with the file name, sampling rate, and EEG net type to the mat\_file\_info\_table in the user\_inputs folder, and re-save the table (and only the table!). Files that do not exist in BOTH the source directory and the mat\_file\_info\_table will NOT be run through the pipeline.

Generating a mat\_file\_info\_table:

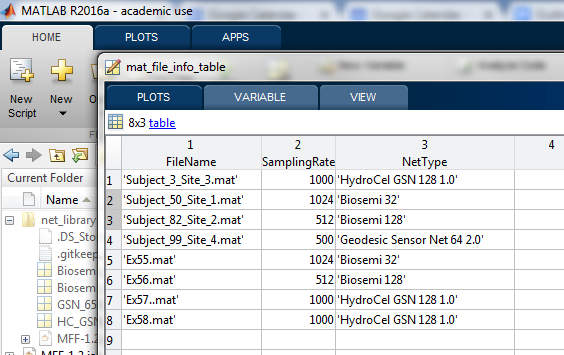
In the example below, the user has 4 source files exported to Matlab, all from different sites and collected with different nets. The user has placed them all in the source directory example\_data:



In Matlab the user has then navigated to /user\_inputs/mat\_file\_info\_table and opened the file by double clicking, before opening the mat\_file\_info\_table variable from the Matlab workspace.



The user has then entered the filename, the sampling rate, and the net type for each of these files. Each net name should match the full name of one of the options in the table under reference\_data/net\_library\_options.mat exactly, unless a new net type is being introduced. In that case, the user can use a consistent, new net name for all of the relevant files, as they will be prompted to add the net later. Note that having additional listings in the table is fine – only files listed in the table that are also present in the current source directory will be run.

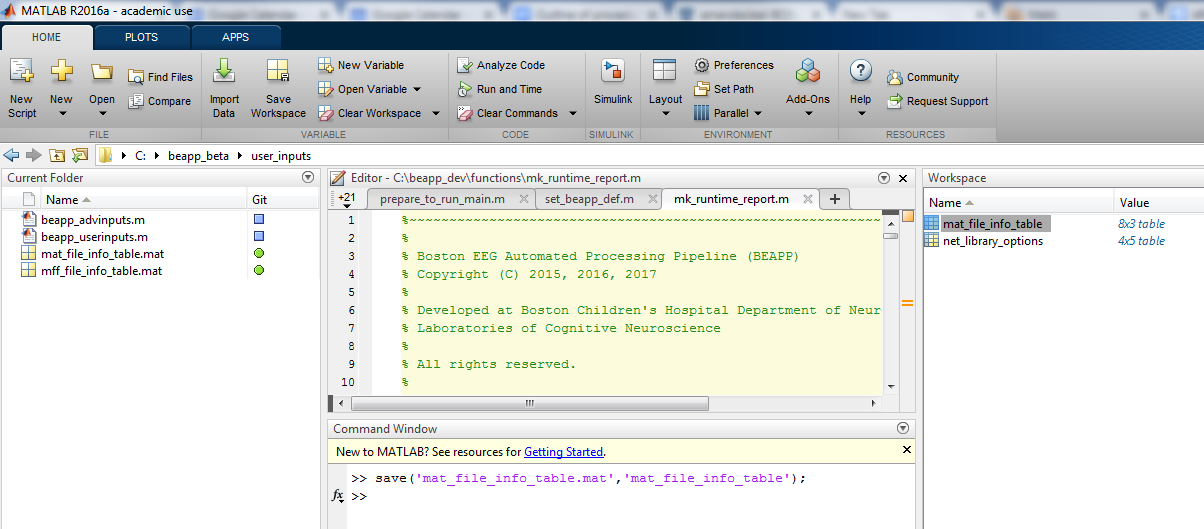


Once the table has been updated, the user should close the variable and save just the mat\_file\_info\_table variable into mat\_file\_info\_table.mat in the user\_inputs folder by navigating to the user\_inputs directory in Matlab, and then entering the following into the Matlab command window:

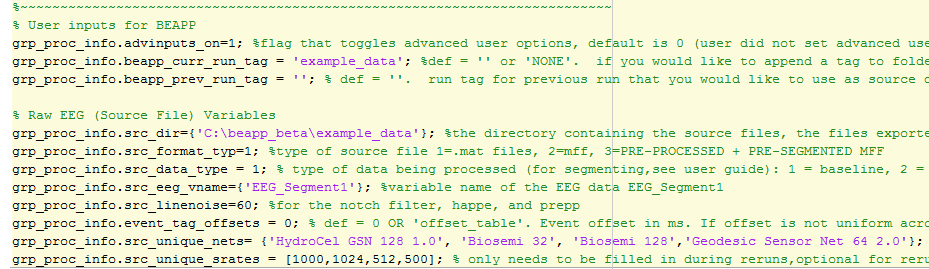
save('mat\_file\_info\_table.mat','mat\_file\_info\_table');

Users wishing to save a copy of the table with a different name (for reuse when switching between datsets) should instead enter the command:

save(‘desired\_table\_name.mat’,’mat\_file\_info\_table’);



**Note:** An example script for generating this table automatically can be found in /reference\_data/example\_scripts under generate\_mat\_file\_info\_table.m . Users are encouraged to modify this script to generate this table for their own data.

The EEG variable name for the example files is EEG\_Segment1. The user will then set the source variables in the user inputs as below: 

The user can then set their other settings, navigate to the BEAPP directory in Matlab, and then enter beapp\_main to run the pipeline.

Note: Users wishing to generate a rerun\_fselect\_table can use the same process, selecting the rerun\_fselect\_table.mat file in the user\_inputs, entering appropriate file information , and saving the table as a variable, rerun\_fselect\_table, inside rerun\_fselect\_table.mat. As with the mat\_file\_info\_table, users may also save the table in an alternative location. This can also be done for event -tagged data.

**Running BEAPP With Differently Structured Data**

**Running Standard Baseline Files**

To run baseline files that contain only baseline EEG data (e.g files that have had relevant sections pulled out in Netstation), set grp\_proc\_info.src\_ data\_type =1 in the user settings. If you have an MFF file where some epochs are entirely baseline and some are event related, you can also run the baseline epochs by setting grp\_proc\_info.src\_data\_type = 1 and selecting the desired epochs in grp\_proc\_info.epoch\_inds\_to\_process.

**Running Files That Contain Baseline Data Between Event Tags or Timepoints**

Beta: Functionality and instructions to come

**Running Files That Contain Alternating or Recurring Sections of Baseline Data (e.g. eyes open, eyes closed)**

Beta: Functionality and instructions to come

**Running Files With Baseline Data With Time-Locked Information (e.g trial-based behavioral coding)**

Data can be run as event-tagged, despite being baseline. See section on running event-tagged data below.

**Running Event- Tagged Data**

To run mff files containing event related data, set grp\_proc\_info.src\_data\_type = 2.

You’ll also need to set stimulus onset, condition name, and condition cell number information for the events you’d like to analyze. The outputs for each condition are separate, so the events don’t need to be from the same experiment, provided they’re in the same file. Additionally, this event related information is first used in the segmentation module, so if you’d like to change the subset of events you’d like to analyze, you can alter these settings and then rerun BEAPP starting at the segmentation module. The events selected at segmentation are automatically the ones used in the output modules.

The standard event related inputs are:

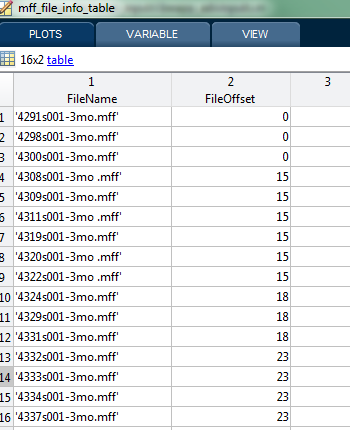
|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.evt\_win\_start | Segment start (in seconds) for event-related segments, relative to event of interest | = - .2000;  = 0.5; | If advanced inputs not used, also used for ITPC |
| grp\_proc\_info.evt\_win\_end | Segment end (in seconds) for event-related segments, relative to event of interest | = -0.1  = 1; | If advanced inputs not used, also used for ITPC |
| grp\_proc\_info.beapp\_event\_code\_onset\_strs | List of strings for target event tags to be analyzed | = {‘stm+’};  ={‘faceon’, ‘houseon’}; |  |
| grp\_proc\_info.beapp\_event\_eprime\_values.condition\_names | Desired condition names for event cell codes (see next line) | ={'Standard', 'Non\_Native', Native}; | Order must match cell codes |
| grp\_proc\_info.beapp\_event\_eprime\_values.event\_codes(:,1)  grp\_proc\_info.beapp\_event\_eprime\_values.event\_codes(:,2)  grp\_proc\_info.beapp\_event\_eprime\_values.event\_codes(:,3) | The cell numbers from E-Prime that correspond to a given condition for event tags. | = [1,2,3];  =[10,12,13];  =[11,12,13]; |  |

grp\_proc\_info.beapp\_event\_eprime\_values.condition\_names is the list of names you’d like to be given to events with the corresponding cell codes listed in grp\_proc\_info.beapp\_event\_eprime\_values.event\_codes. This list of names does not need to match whatever name was given in the E-Prime file**, but the order of the condition names and the cell names must match exactly.** If the cell numbers used for each condition have changed or are different at different sites, users can add additional rows of cell numbers that correspond to the appropriate conditions. **BEAPP will choose the set of cell numbers that has the most overlap** with the contents of a file as the assumed condition information. grp\_proc\_info.beapp\_event\_code\_onset\_strs should include all possible stimulus onset tags for the conditions selected, but the order need not match the other inputs.

**Running Event Related Data, continued:**

**File Event Tag Offsets:**

In order to adjust for timing offsets in event tagging in NetStation/E-Prime, users have two options. If all files/events in the dataset have the same offset, the user may set grp\_proc\_info.event\_tag\_offsets to the offset in ms. This offset will usually be positive (ePrime offsets typically cause event markers to appear before a participant is actually presented with a stimulus due to delays caused by presentation hardware), but in some cases may be negative. If files have different offsets, users will need to set grp\_proc\_info.event\_tag\_offsets = ‘offset\_table’, and then load and fill out the mff\_file\_info\_table in the user\_inputs folder to include the file name and sampling rate for each mff file (for more detailed instructions on this process, see the example for the mat\_file\_info\_table under .Mat exports in Running BEAPP with Different Source File Formats). After filling out the table, save only the mff\_file\_info\_table variable to the mff\_file\_info\_table.mat file in user\_inputs (as shown in the .mat export example).



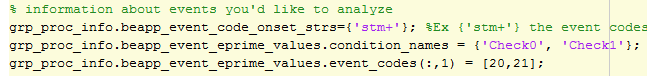
**Note:** An example script for generating this table automatically can be found in /reference\_data/example\_scripts under generate\_mff\_file\_info\_table.m . Users are encouraged to modify this script to generate this table for their own data.

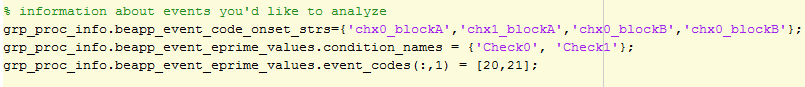
**Notes for Event Processing in BEAPP:**

1. Event offsets are set during format, and are expected to be consistent within a file. They are updated during resampling if necessary.
2. BEAPP selects the cell code/ condition set provided by the user that has the best fit for the events in the file. BEAPP will skip files if the events present don’t overlap with the provided cell codes. If 2 sets of cell codes have an equal amount of overlap with the events present in a file, BEAPP will choose the first one and give a warning.
3. Running all events for all experiments in a file at once can be convenient, but because BEAPP will store segments (and later output metrics) for all selected conditions in each file, running too many conditions at once can dramatically increase the size of segmentation output files and of output files.
4. Information about the events within a file can be found in file\_proc\_info.evt\_info in the appropriate epoch. Sample numbers will correspond to the current sampling rate for a file, and will change during the resampling module to match the new sampling rate
5. By default, segments with the same event tag are collapsed across epochs.
6. Beta: At present, BEAPP does not have the option to have one condition designated by different cell numbers, although there are workarounds if needed
7. Beta: At present, BEAPP can only run event related data from EGI MFF files collected with E-Prime.

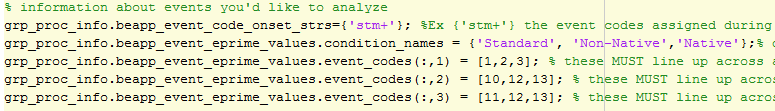
**Examples of Event Related User Settings:**

Example 1: User is analyzing 2 conditions for one task. Stimulus onset was marked using the same tag for both conditions.

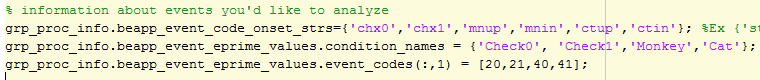


Example 2: User is analyzing 2 conditions for one task. Stimulus onset was marked using different tags depending on the condition and block, but corresponding events have the same cell number. The user would like to analyze all Check0 trials together independent of block, and the same for Check1.

Example 3: User is analyzing 3 conditions for one task. The stimulus onset for all of the conditions was marked using the same stm+ tag, but the cell numbers used to represent each condition were changed during the course of the study.



Example 4: User is analyzing 2 conditions for a checkerboard task and 2 conditions for a task that displays monkeys and cats (here, mn and ct, either in inverted or upright position). Each condition only has one possible associated cell number.



**Behavioral Coding Information**

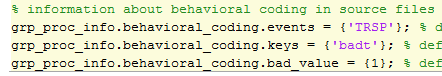
Beta: Behavioral coding information can be read in from files and can be seen in file\_proc\_info.evt\_info, but excluding segments based on behavioral coding is not yet an option. Instructions for future usage below.

If behavioral coding information is encoded in the event tags, users have the option to add the event tags, event keys, and key values (strings or numbers) that contain information about trial quality. These should line up exactly across the three input lists. If any of the relevant keys is marked “bad” for a trial, that trial will be excluded (not true in BEAPP Beta).

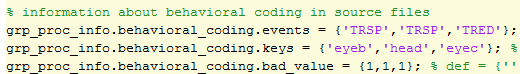
|  |  |  |  |
| --- | --- | --- | --- |
| **Variables/ Inputs** | **Description** | **Examples** | **Notes** |
| grp\_proc\_info.behavioral\_coding.events | List of events with behavioral coding. Default = {‘’}; | ={'TRSP'};  ={‘TRSP’,’TRSP2’}; | Optional |
| grp\_proc\_info.behavioral\_coding.keys | List of keys with behavioral coding in events. | **=** {'badt'};  ={‘badtf’,’badth’}; | Optional |
| grp\_proc\_info.behavioral\_coding.bad\_value | Value that marks behavioral coding as bad | ={’’};  ={1}; | optional |

**Behavioral Coding Information, continued:**

Example 1: Bad trial information contained in the TRSP tag under the badt key, with bad value 1.



Example 2: Bad trial information contained in the TRSP and TRED tags under the eyeb, head, and eyec keys, with bad value 1. If any of the relevant keys is marked “bad” for a trial, that trial will be marked bad.

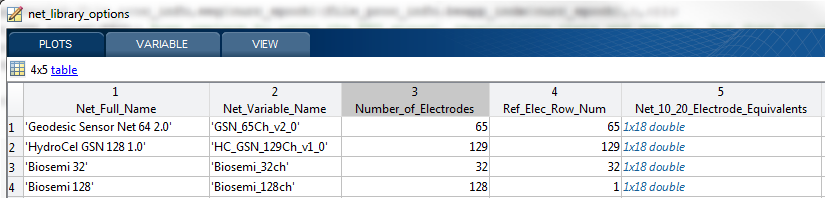


**DIN Tag Information**

DIN tags that have been pre-matched can be treated as regular events, following the instructions above. Beta: DIN matching functionality and instructions to come

**BEAPP Net Library**

The BEAPP net library contains .mat files with the coordinate information for each of the nets that have previously been used to run BEAPP on a given computer. You can see which nets are currently in the net library by opening the catalog. This is done by double clicking the net\_library\_options\_table.mat file in the reference\_data folder from the Matlab viewer, and clicking finish.



In the net library catalog, the Net\_Full\_Name is the exact name of the net from the source file (MFF net listing, for example). For MFF files, this can be found in Netstation in Edit> File Info >Sensor Layout. For mat exports, this should be the exact name you intend to use in your mat\_file\_info\_table. The Net\_Variable\_Name is a user-defined abbreviation. If for some reason the same net can have different names in your source files, you can add multiple listings for the net with different Net\_Full\_Names, but the same Net\_Variable\_Name.

**Adding New Nets to the BEAPP Library**

If the data being run was collected with a net type that has not been used to run BEAPP on your computer before, you will need to add that net to the net library. This can either be done at the run start or as BEAPP detects files with the new net type. The latter will require the user to pay attention during the run, as BEAPP will prompt the user and pause until the net is added. You only need to add a given net once on each new computer, and users can transfer their net libraries and net library catalogs to new computers to avoid reentering the information.

**Adding New Nets to Library During a Run (not recommended):**

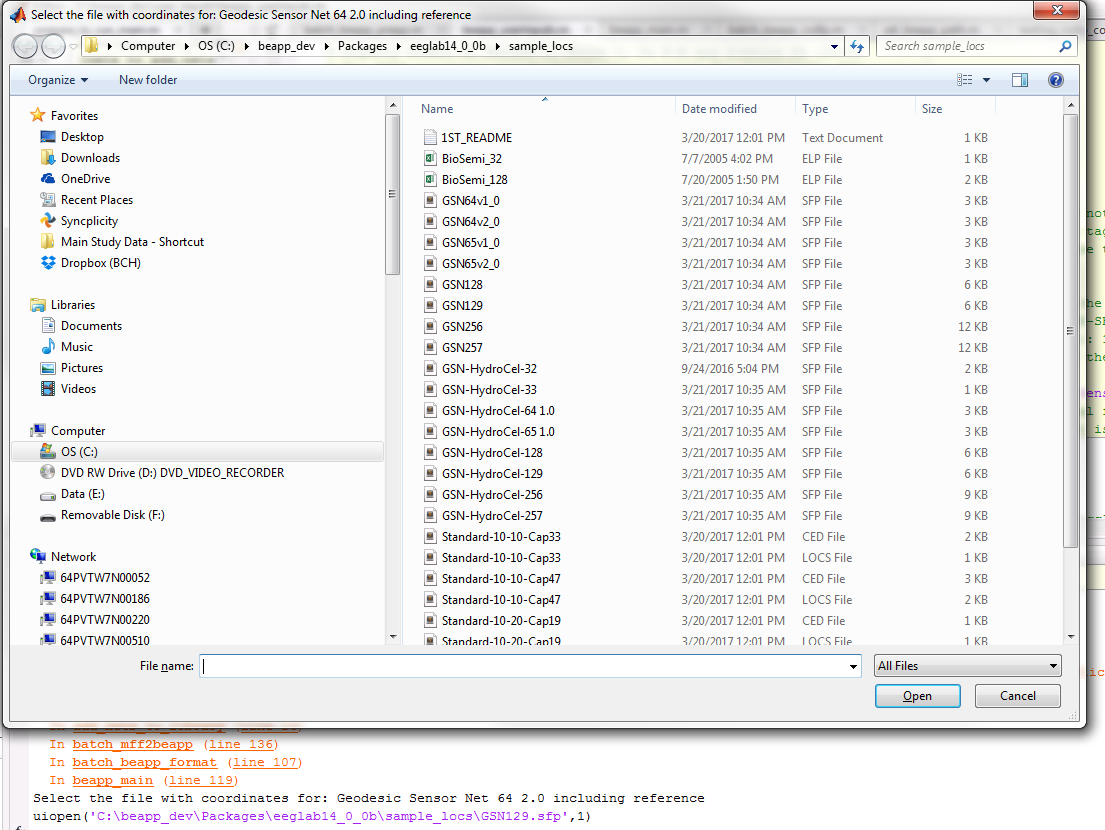
Adding nets during a run is only recommended if you do not know the exact names of your nets as they exist in the file. If BEAPP detects a new net during a run, the user will be prompted to select the file containing the coordinates for that net type. Nets are then added as below.

**Adding New Nets to Library At Run Start (recommended):**

To add a new net before a run, you will need the exact names of the nets you would like to add as it is listed in the source files. Enter the list of nets in the dataset (all, whether they are in the library or not) into the grp\_proc\_info.src\_unique\_nets variable in the user inputs. For example if you’d like to add an EGI HCGSN 129 channel net and an EGI GSN 65 channel net to your library, but have already added 2 Biosemi nets that are also present in your dataset, you would set grp\_proc\_info.src\_unique\_nets = {'HydroCel GSN 128 1.0', 'Biosemi 32', 'Biosemi 128', ‘Geodesic Sensor Net 64 2.0’}; in the user inputs.

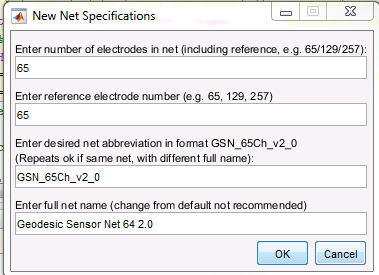
**Adding New Nets to Library At Run Start, continued:**

Once you run beapp\_main, BEAPP will automatically prompt you to select the file with the relevant coordinate information for the new nets, as shown below:



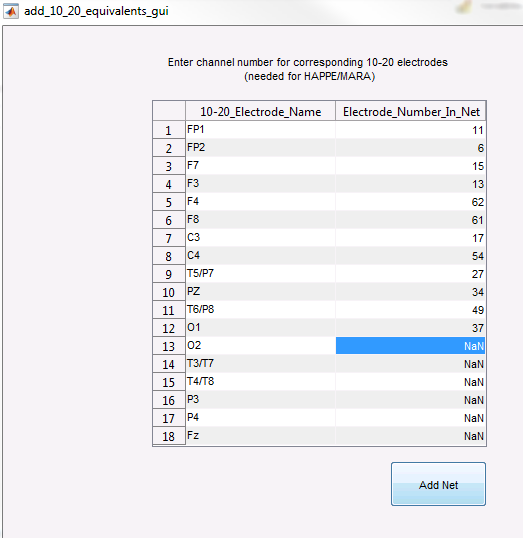
By default, BEAPP will open to the EEGLAB sample\_locs directory, but you can navigate anywhere to select your relevant channel locations files. BEAPP uses the eeglab pop\_readlocs function to read channel positions, so channel positions can be in any format supported by EEGLAB.

Select the appropriate file (including reference) and click open. You will then be prompted to enter some additional information about the net:



Enter the information and click ok. While changing the full net name is not recommended, we recommend entering the net abbreviation name in the format shown in the image for consistency. Click ok when you are finished.

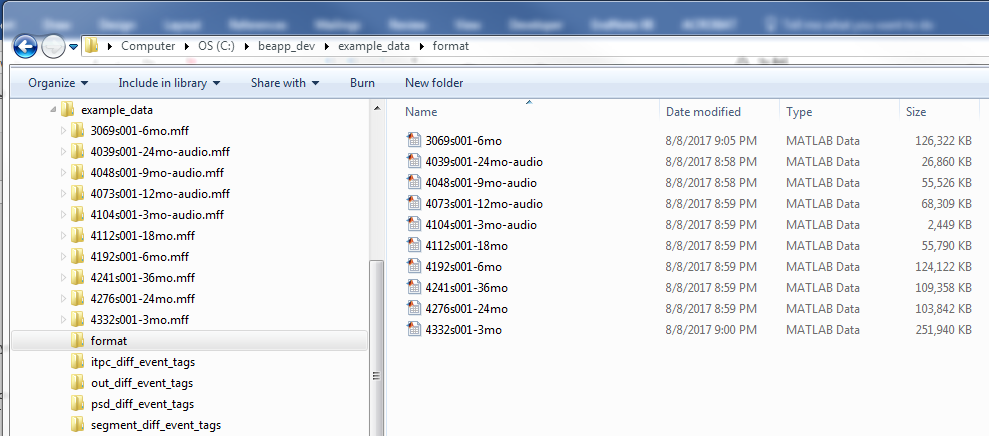
Finally, you will be prompted to enter which channel numbers (rows in the data) correspond to each of the 10 20 electrodes. Users not intending to use HAPPE can leave these as NaNs and select OK, but it’s recommended to enter these correctly, as you will need this information if you ever decide you would like to run HAPPE or another 10-20 based analysis using this net type.



**BEAPP Outputs and Reporting**

**Outputs**

Outputs for each module in the pipeline will be stored as .mat files in the corresponding directory named for the module (see example below for format module outputs). Each file will contain a file\_proc\_info structure variable within them with processing information about that file, and a second variable containing the eeg data at the current stage of the pipeline. Files with no data left at the end of a module or with invalid source data will not be saved in a module’s output directory.



**File Processing Information**

File processing information for each BEAPP output file can be found in the file\_proc\_info variable. To access this information, double click on the data file in the appropriate module folder to load it into Matlab, and then double click on the file\_proc\_info variable in the workspace. Each variable stored in file\_proc\_info should have a prefix related to its function. Generally:

|  |  |  |
| --- | --- | --- |
| **Prefix** | **Usage** | **File\_proc\_info Field Example** |
| .beapp | current file information used | .beapp\_srate (the current file srate) |
| .src | information related to the source file format | .src\_srate (original srate) |
| .evt | information related to file events | .evt\_info (struct containing event tag information for a file) |
| .hist | File/run history information | .hist\_run\_table (information about modules applied to file) |
| .epoch | Information about recording periods (this will change to .rec\_period in future versions) | .src\_epoch\_end\_times (recording period end times in MFF files) |
| .net | Information about nets used | .net\_typ (file net name) |
| .ref | Information about reference data/resources |  |
| .seg | Information about segments created outside BEAPP | .seg\_info (struct with segment information generated in pre-segmented MFF files) |
| .grp | Group information that does not change and needs to be stored locally | .grp\_wide\_possible\_  cond\_names\_at\_segmentation |

Depending on the processing stage and the source file type, you will see some or all of the variables below (note: recording periods are listed as epochs for the time being):

* Current file information used (.beapp variables). Here, epochs are recording periods.

|  |  |  |
| --- | --- | --- |
| **File\_proc\_info Variable Name** | **Description** | **Notes** |
| file\_proc\_info.beapp\_bad\_chans | List of bad channels detected in BEAPP for each epoch analyzed from source file | List within each cell corresponds to each epoch |
| file\_proc\_info.beapp\_fname | BEAPP filename for this file |  |
| file\_proc\_info.beapp\_indx | List of channels BEAPP is using for analysis at the current stage | Will change depending on interpolation/removal/selection of channels |
| file\_proc\_info.beapp\_nchans\_used | Number of channels being used by BEAPP in each epoch analyzed | Will change depending on interpolation/removal/selection of channels |
| file\_proc\_info.beapp\_num\_epochs | The number of epochs from the source file that BEAPP is analyzing | Impacted by user selection of epochs in inputs |
| file\_proc\_info.beapp\_srate | The current sampling rate for the file | Will differ from src\_srate if file has been resampled |
| File\_proc\_info.beapp\_win\_size\_in\_samps | Window size in samples (using beapp\_srate) | Only output for baseline files |

* Epoch/recording period information (.epoch variables)

|  |  |  |
| --- | --- | --- |
| **File\_proc\_info Variable Name** | **Description** | **Notes** |
| file\_proc\_info.epoch\_inds\_to\_process | Index of epochs in source file that were chosen for processing in this file |  |

* Epoch/recording period information (.evt variables)

|  |  |  |
| --- | --- | --- |
| **File\_proc\_info Variable Name** | **Description** | **Notes** |
| File\_proc\_info.evt\_conditions\_being\_analyzed | Table with the event codes, user given condition name, and native file condition name for each condition analyzed in that file | May be a subset of events in source file – depends on user event selection |
| File\_proc\_info.evt\_header\_tag\_information | Information extracted from file event track header (MFF files) | May be empty |

* Epoch/recording period information (.evt variables), continued

|  |  |  |
| --- | --- | --- |
| **File\_proc\_info Variable Name** | **Description** | **Notes** |
| File\_proc\_info.evt\_info | Information about all event tags present in file: time, sample number, epoch, label, cell number, bad trial coding | Separated by source file epoch (events from each epoch within each cell) |
| File\_proc\_info.evt\_seg\_win\_evt\_ind | Index of event tag within original segmentation window | Used to find edges of analysis window |

* Group wide information that cannot change (.grp variables)

|  |  |  |
| --- | --- | --- |
| **File\_proc\_info Variable Name** | **Description** | **Notes** |
| File\_proc\_info.grp\_wide\_possible\_cond\_names\_at\_segmentation | Ordered condition names at segmentation, determines order of segments in eeg\_w in all files produced in that run |  |

* ICA module stats:

|  |  |  |
| --- | --- | --- |
| **File\_proc\_info Variable Name** | **Description** | **Notes** |
| File\_proc\_info.ica\_stats | Contains channel and IC information for this file if HAPPE was run. Same information given for files in the HAPPE reports | Only present if ICA has been run |

* File history variables (.hist variables)

|  |  |  |
| --- | --- | --- |
| **File\_proc\_info Variable Name** | | **Description** |
| File\_proc\_info.hist\_run\_table | | Contains information on which modules have been applied to this file, when the run where each module was applied was started, how long the file took in each module, and what tag name was used |
| File\_proc\_info.hist\_run\_tag | Date and time that run for this file was started | |

* File net information (.net variables)

|  |  |  |
| --- | --- | --- |
| **File\_proc\_info Variable Name** | **Description** | **Notes** |
| File\_proc\_info.net\_10\_20\_elecs | Electrodes corresponding to 10\_20 electrodes for this net | Information set and accessible in the net library catalog |
| File\_proc\_info.net\_happe\_additional\_channel\_lbls | List of additional channels (beyond 10-20s) analyzed using HAPPE for this net | Set by user |
| File\_proc\_info.ref\_elec\_rnum | Index of reference electrode for this net | Information set and accessible in the net library catalog |
| file\_proc\_info.net\_typ | File net type name |  |
| file\_proc\_info.net\_vstruct | Net channel positions for this file |  |

* File pre-generated segment information (.seg variables). Note that this may appear in files with large “segments” used to pull paradigms out of a recording in Netstation, even if the files were not pre-segmented (.src\_format\_typ = 3) .MFFs

|  |  |  |
| --- | --- | --- |
| File\_proc\_info.seg\_info | Information about all the segments present in file: time, label, cell number, bad trial coding, hand editing information | Only present in files read into BEAPP after segmentation |
| File\_proc\_info.seg\_tasks | Tasks present in pre-generated segments |  |

* Source file information (.src variables). Some of the variables unlikely to be useful to users have been excluded from this list.

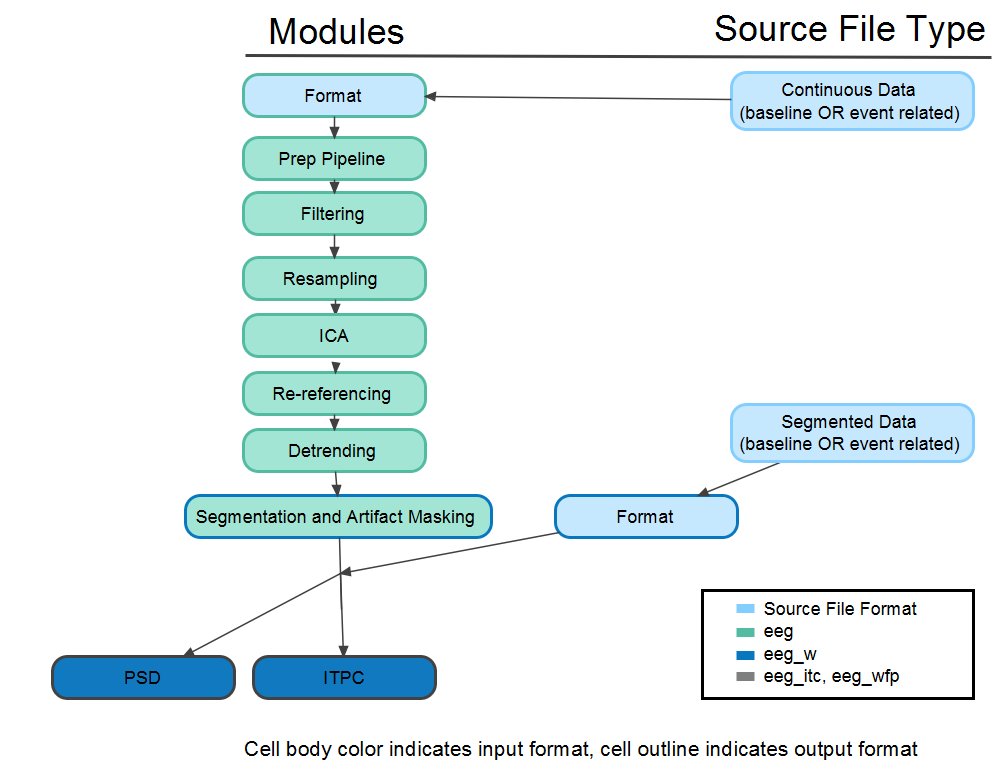
|  |  |  |
| --- | --- | --- |
| **File\_proc\_info Variable Name** | **Description** | **Notes** |
| file\_proc\_info.src\_amp\_serial | Amplifier serial number |  |
| file\_proc\_info.src\_amp\_serial | Amplifier type |  |
| file\_proc\_info.src\_epoch\_nsamps | Number of samples (using source srate) in each epoch in original file |  |
| file\_proc\_info.src\_file\_offset\_in\_ms | User input, file offset in ms |  |
| file\_proc\_info.src\_fname | Source filename for this file |  |

* Source file information (.src variables), continued. Some of the variables unlikely to be useful to users have been excluded from this list.

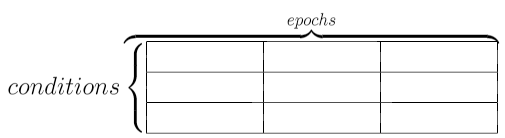
|  |  |  |
| --- | --- | --- |
| **File\_proc\_info Variable Name** | **Description** | **Notes** |
| File\_proc\_info.src\_nchan | Source linenoise frequency |  |
| File\_proc\_info.src\_nchan | Original number of channels for this file |  |
| File\_proc\_info.src\_num\_epochs | Number of recording periods in source file | Different from beapp\_num\_epochs depending on user selection of epochs in inputs |
| File\_proc\_info.src\_record\_start\_day | Date file recorded |  |
| File\_proc\_info.src\_record\_start\_time | Time file recorded |  |
| File\_proc\_info.src\_srate | Original source file sampling rate | Will differ from beapp\_srate if file has been resampled |
| File\_proc\_info.src\_subject\_id | Subject ID, if provided in source file |  |

**Individual EEG Data/ EEG Analysis Outputs**

All BEAPP EEG outputs are Matlab cell arrays with variable names that are dependent on the stage of the pipeline. The variable name for the eeg output during the continuous data stage is **eeg** (all lowercase) and is **eeg\_w** after segmentation. Outputs from the analysis/output metric stage of the pipeline vary for each module (ex: eeg\_itc, eeg\_wfp).

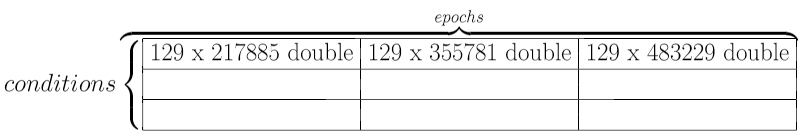


Each cell in the cell array contains a numerical 2D or 3D array with the EEG data, dependent on the pipeline stage. In general, BEAPP EEG output cell arrays are organized by splitting data horizontally by epoch before segmentation occurs, and then vertically by condition after segmentation, as below:



**Continuous/Not Yet Segmented EEG Outputs (eeg)**

All individual output files from modules that produce continuous data (all modules before segmentation) will contain the file\_proc\_info variable and a second variable, eeg. The eeg variable is a cell array, with each cell representing an epoch from the original file, and containing the 2D channel x sample array with data from the epoch. In the example below, 3 experiments were run during the participant’s EEG session, each with its own epoch. Data collected from each of these epochs is placed in its own cell. Users wishing to access data from the first epoch, for example, should load relevant output file, and access eeg{1}.



**Segmented Outputs**

All individual output files from modules that produce segmented data will contain the file\_proc\_info variable and a second variable, eeg\_w. Each cell in the eeg\_w array will contain outputs for a given condition in the form of a 3D channel x sample x trial array. By default, segments with the same event tag are collapsed across epochs. The order of the conditions used to store the cell array can be found in file\_proc\_info.evt\_conditions\_being\_analyzed. In the example below, the user has selected 3 conditions to analyze, each 9090 samples long, with 34, 27, and 51 trials, respectively.



**Analyses/ Output Metric Outputs**

The contents of the output files for analysis modules will depend on the analysis selected, but they all follow the same format as the segmented outputs. Each cell in the eeg\_out array will contain outputs for a given condition in the form of a 3D array (channel x sample x trial).



**Reports**

**Run Reports**

In the current version of BEAPP, a very rudimentary report of command line outputs (file specific warnings, for example) can be found in the out directory for a run. Additionally, a Matlab structure containing all the user settings and dataset wide information collected during the run can be found in the Run\_Report\_Variables\_and\_Settings.mat file in the out directory. Formatted reporting will be added in future BEAPP versions.

**Analysis/ Module Reports**

The PREP module will automatically generate a list of files that failed in PREP. Users have the option to generate reports containing basic information for each file in the dataset in HAPPE, PREP, and all output modules (ITPC, PSD, etc.) using the user inputs. These reports are tables containing basic file information for each file, along with the relevant outputs for each file in the dataset. A different table will be generated for each event type/condition, and for each kind of output metric selected (e.g. raw power vs. log power). Files that did not survive the pipeline (because of a lack of usable segments, for example) will have NaNs listed in place of an output metric. For further information about the HAPPE outputs, please see the companion paper detailing HAPPE by Gabard-Durnam and colleagues in this issue.

.mat file versions of the output metrics for the dataset are also saved in the out folder for a run, for users who would like to continue processing the data in Matlab. These are stored in cell arrays named report\_values in .mat files named module\_output.mat (ex psd\_output.mat) following the BEAPP epoch x condition output format. Each cell contains a 3D array, which is subject x observation x output metric. The observation names are listed in hdr\_out and the output metric names are listed in tname out.

**Additional Information**

BEAPP integrates the following preexisting software packages. These are provided in Packages.zip in the BEAPP repository

EEGLab 14.0.0b

EEGLAB plugins:

* Dipfit 2.3
* Firfilt1.6.2
* MARA
* PrepPipeline v0.52
* Cleanline
* Fieldtrip (only if using topoplots)

CSD Toolbox