HEDTools Plugin User Manual

October 31, 2016

Table of Contents

Table of Figures

Table of Tables

1. Getting Started with HEDTools

1.1 Overview

HEDTools is an EEGLAB plugin designed to help users annotate and validate events or other data elements using a predefined, but extensible, hierarchically structured annotation language. The input to the system consists of two parts: a list of items to be annotated or validated and an annotation hierarchy. In the case of EEG, users annotate the events that occur during an EEG experiment using the HED 2 hierarchical event description language as the vocabulary. Events that have been tagged from a past experiment can be validated to make sure that they are present in and meet the requirements of HED 3.

1.2 Requirements

HEDTools is dependent on MATLAB and EEGLAB. Please use the most current version of EEGLAB.

1.3 Installation

To use *HEDTools* unzip *HEDTools1.0.0.zip* under the *EEGLABPlugin* directory so that the *HEDTools1.0.0* directory is directly under the EEGLAB *plugins* directory. When you run EEGLAB, the paths will automatically be set up.

2. Tagging Data

The *HEDTools* plugin adds three additional menu items to the EEGLAB menus that are associated with annotating data and they are the following: *Tag current dataset* (under the *Edit* menu), *Tag study* (under the *Tag files* submenu under the *File* menu), and *Tag directory* (under the *Tag files* submenu under the *File* menu).

2.1 Tagging a single dataset

First load a dataset into the workspace by clicking the *Load existing dataset* menu item under the *File* menu. To tag the dataset click the *Tag current dataset* menu item under the *Edit* menu which is illustrated below in Figure 1.

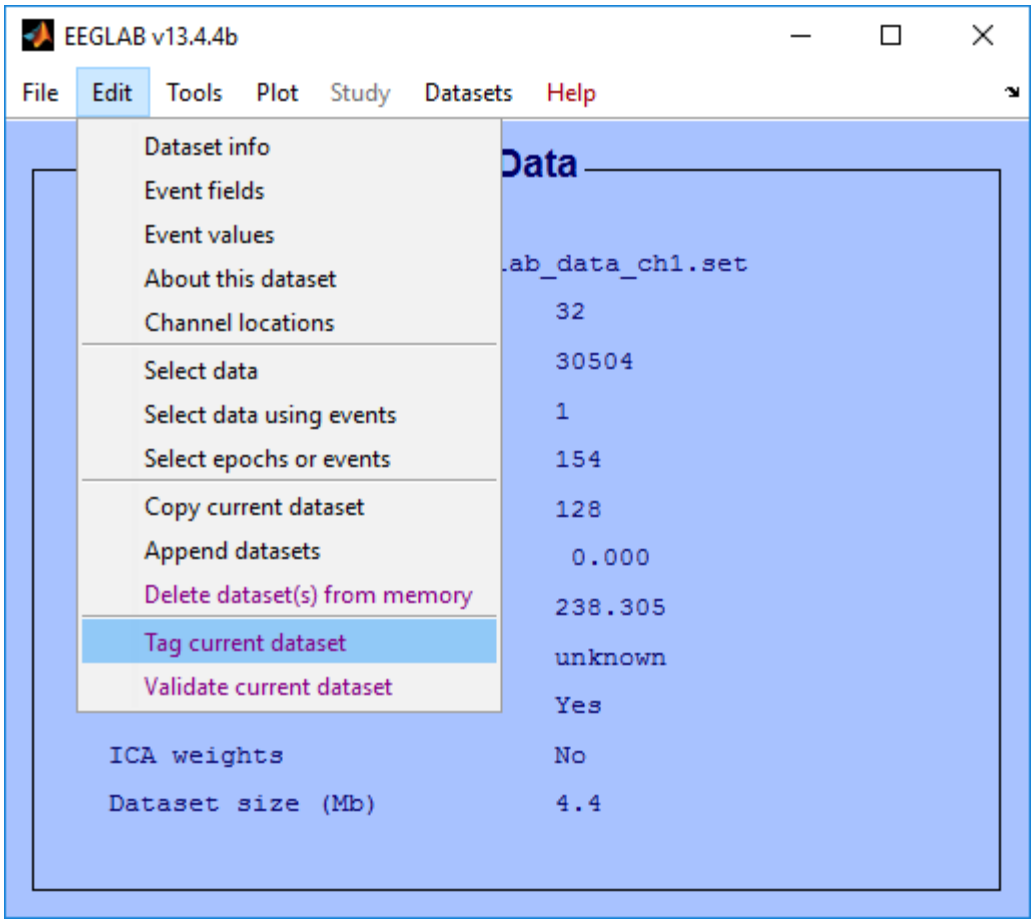

Figure 1. Tagging the current dataset from the EEGLAB Edit menu

The *Tag current dataset* menu item executes the *pop_tageeg* function which brings up a menu for specifying options for tagging. Any tag information in the dataset is extracted and this information is used to list any existing tags. Each unique field in the *EEG.event* structure is tagged independently. Normally, you would just tag the values that appear in *EEG.event.type* field. However, if there are additional fields in the *EEG.event* structure to better distinguish subcategories, the events are defined by the unique combinations of values in the fields.

Depending on the options set in the *pop_tageeg* menu you are allowed to specify which fields to tag and which fields to exclude from tagging through a menu. This method requires that your labeling scheme be orthogonal --- that is, it assumes each field can be tagged separately and the tags from the fields are combined for an event.

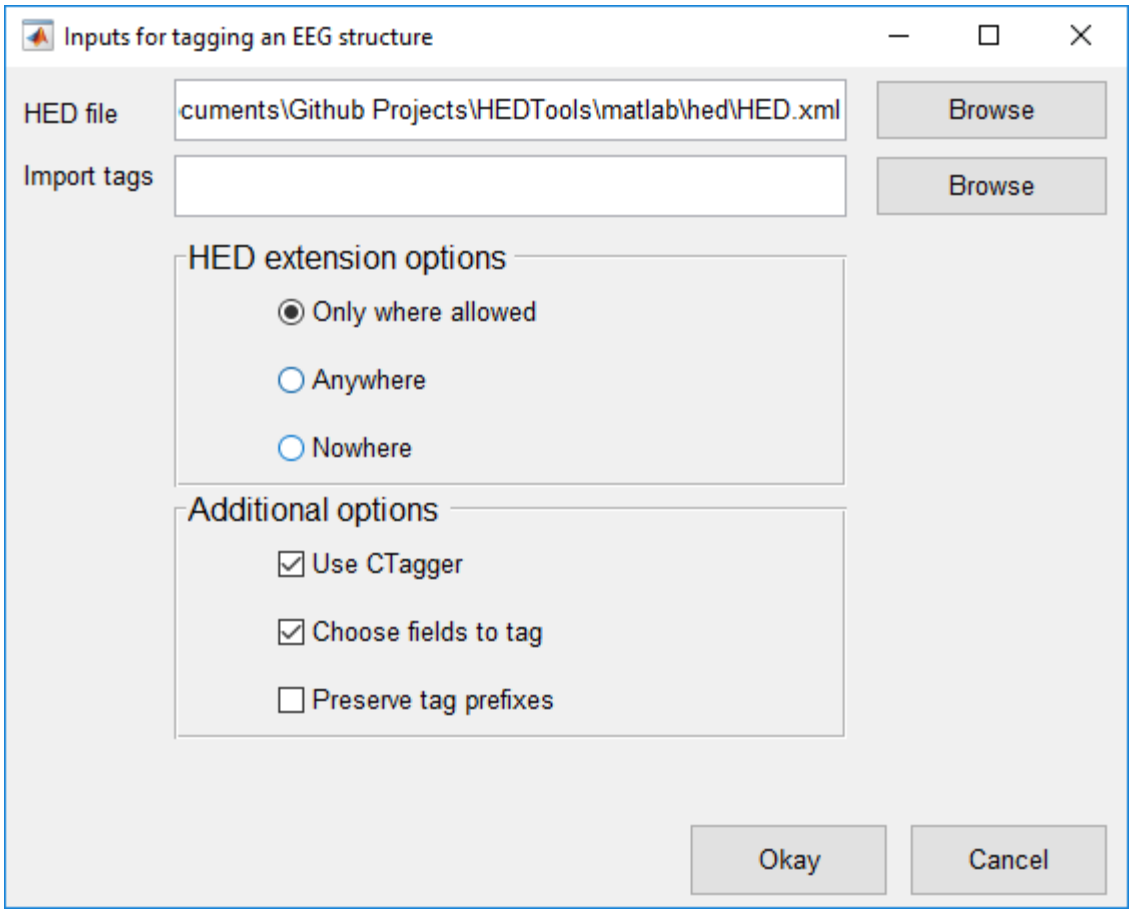

Figure 2. pop_tageeg menu

In Figure 2 above, the top section of the menu allows you to browse for a HED *.xml* file and a *.mat* file containing tags to import. The HED file by default will be set to the *HED.xml* file found in the *hed* directory. When browsing for an import file only *.mat* files will be considered. The next section allows you to specify where the HED can be extended. These options include:

- Only where allowed - The HED can be extended for leaf tags and tags that have the *extensionAllowed* attribute.
- Anywhere - The HED can be extended for all tags.
- Nowhere – The HED cannot be extended at all even for leaf tags and tags that have the *extensionAllowed* attribute.

The last section allows you to select additional options. These options include:

- Use CTagger to tag each selected field (*Use CTagger)*
- Choose fields to tag through a menu (*Choose fields to tag)*
- List the tags that have the most specific tag starting with that prefix or share the same prefix (*Preserve tag prefixes).*

Once all options are set click the *Okay* button to proceed. If the *Choose fields to tag* checkbox is checked then the following menu below will be presented.

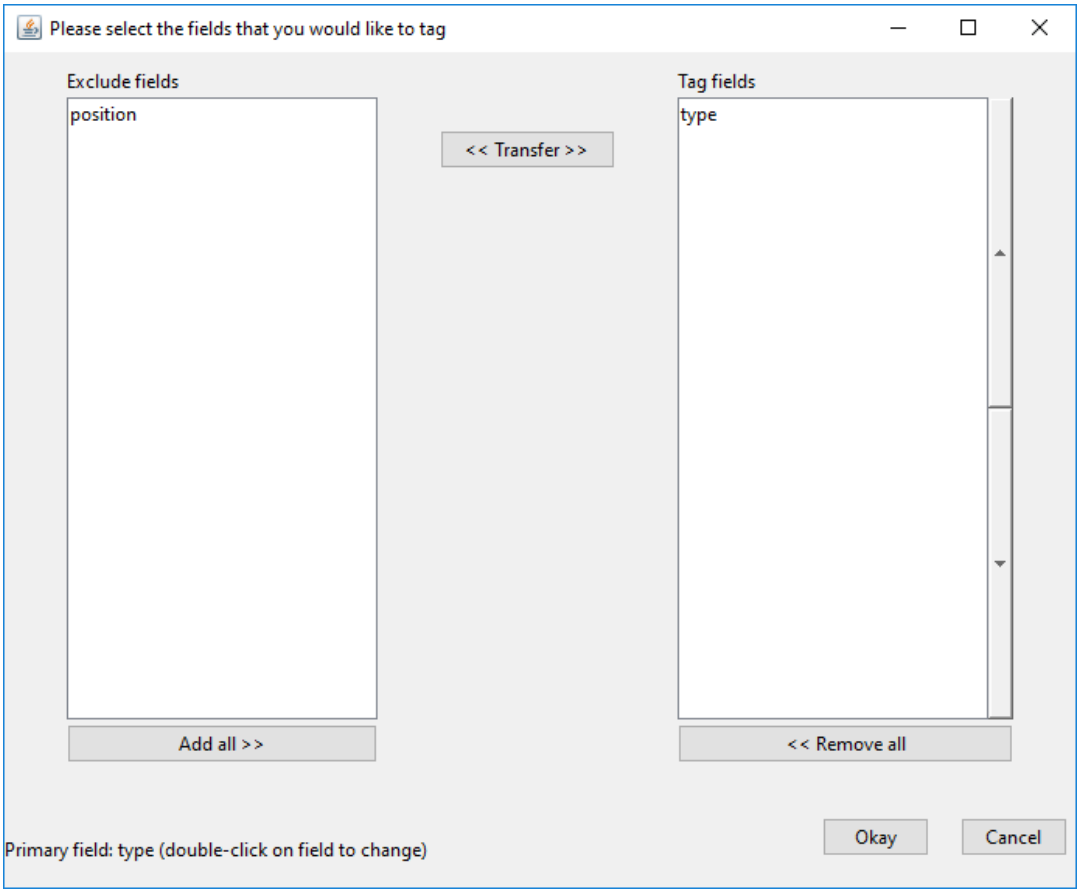

Figure 3. Menu for choosing fields to tag

Figure 3 shows the menu for choosing fields to tag and exclude. If you want to move a field over from one list to the other click on the field and press the *Transfer* button. In addition the left and right arrow buttons on the keyboard can be used in place of the *Transfer* button. Simply press the arrow key that points in the direction to the list that you want to transfer the field to. If you double click on a field then it will be set to the *primary* field. The *primary* field is the field used to specify what kind of event is occurring while the other fields are subfields which are used to specify conditions within the event. The *primary* field requires a label tag (event/label), a category tag (event/category), and a description tag (event/description) for each of its unique values. By default the *primary* field is set to *type.* The up and down arrow buttons next to the *Tag fields* list can be used to specify the order of the fields for tagging. Once the fields have been selected to tag and the *Okay* button is pressed, the tagging application CTagger (in Figure 4) is executed for each field in the tagging list.

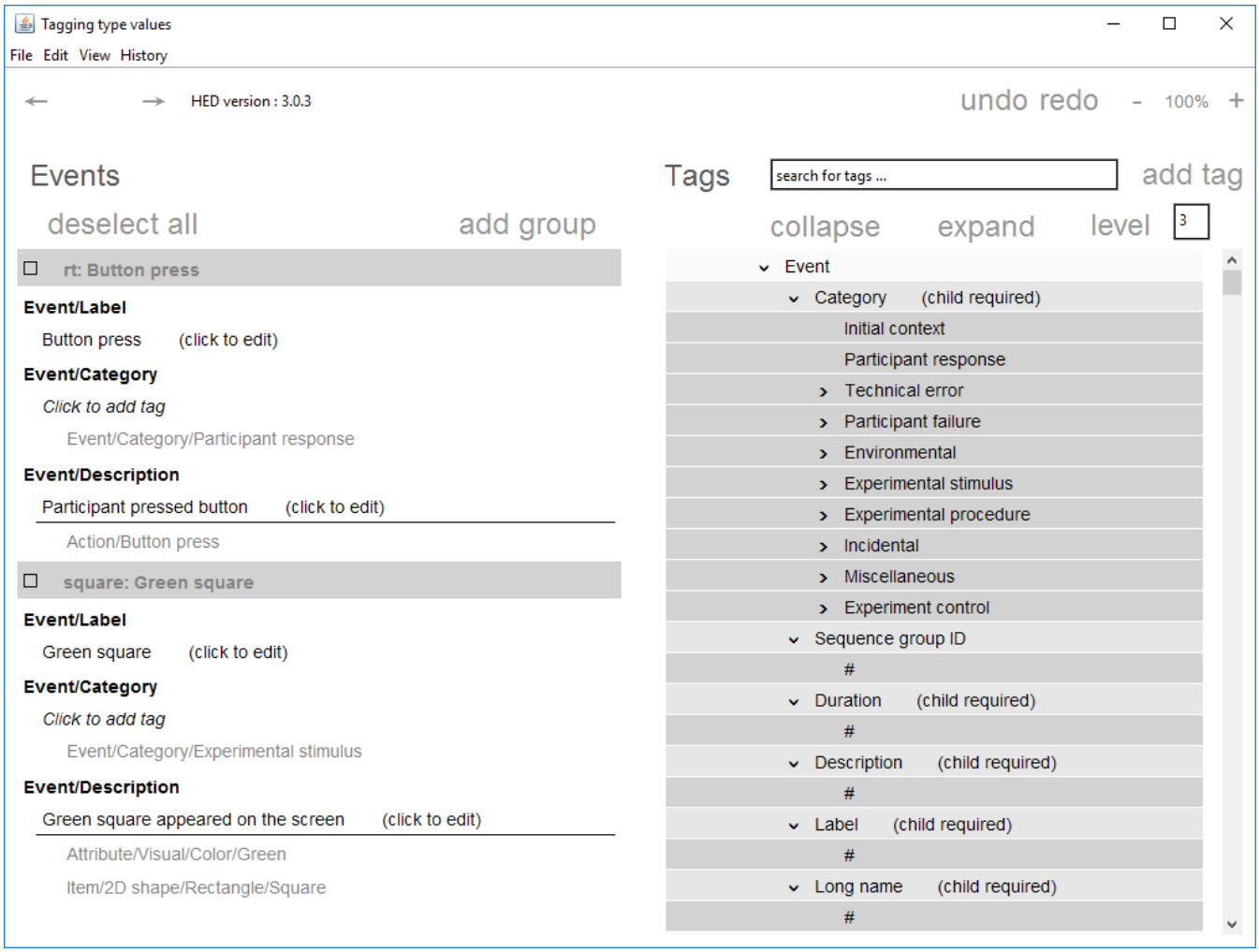

Figure 4. Tagging GUI for the type field

In Figure 4, CTagger allows you to associate tags with each unique value of the current field. Instead of having to choose tags at random, you select from a menu of potential tags organized in a hierarchical format (*HED*) from general to more specific. To tag, select a value by clicking on a checkbox on the left side of the menu. From there select a tag from the HED on the right side of the menu. The tag will then appear underneath the value. To remove a tag right click on it and press *remove.* Based on the *HED extension options*, you can add new tags to the HED. To add a new tag click on a tag that can be extended and click *add* tag. You will be prompted to fill in the tag attribute information. The name is the only field that is required. If this is the first tag you have added then you will be prompted to specify the version number associated with your version of the HED.

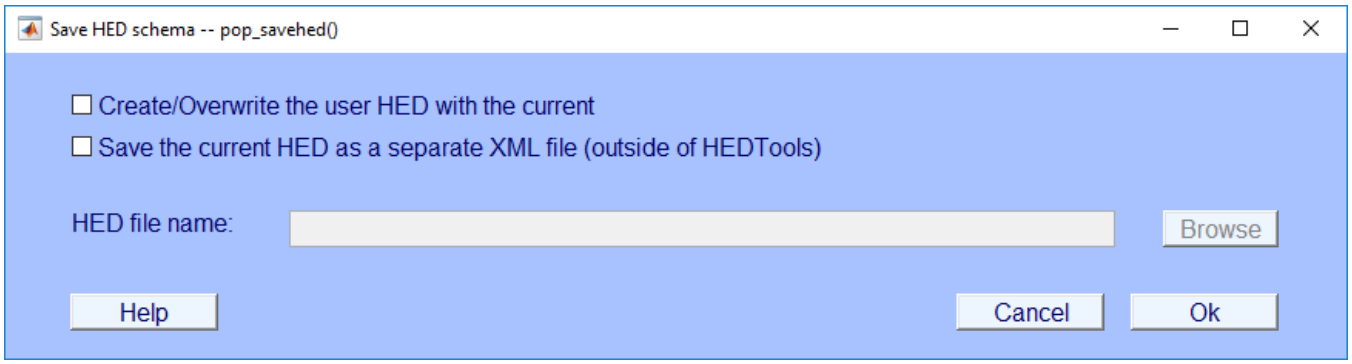

Figure 5. Saving the HED

Figure 5, above will appear if the HED is modified through the CTagger. The first option allows you to save the current HED as the *user* version. The *user* version is strictly reserved for extending the HED. It is saved as *HED_user.xml* in the *hed* directory and is separate from the default version so that it doesn't interfere with fetching the latest version from the repository. The second option allows you to save the current HED as any file name and to any location; preferably outside the HEDTools directory. You may want to reserve copies and keep track of all changes made to the HED overtime. After completion, the following figure will appear.

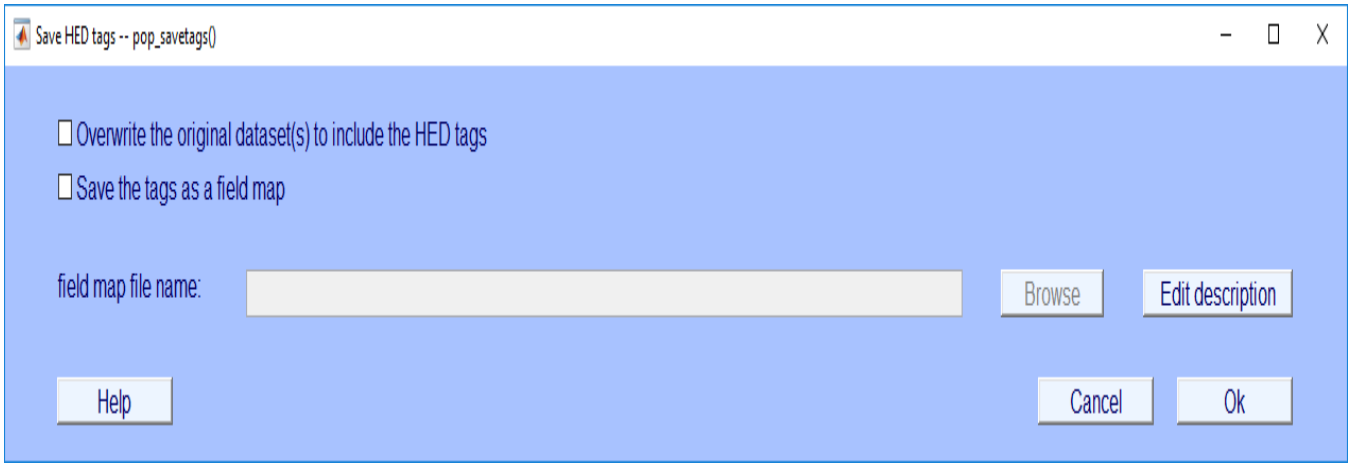

Figure 6. Saving the HED tags to the dataset(s)

In Figure 6, you have two options for saving tags: the first is to overwrite the *.set* dataset(s) to include the HED tags and the second is to save the tags as a field map object as a *.mat* file. Refer to section 5.3 on how tags are saved inside of a dataset and to get a further explanation of what a field map is. The idea behind saving a field map is that you want to use it to tag a different dataset from the same study. Example 2.3 illustrates how to do this. When selecting the second option, the *Browse* button next to *field map file name* is enabled so that you can specify the file name of the field map. The *Edit* description button is not only associated with the field map file saved but the dataset files(s) if you decide to overwrite them. The description will appear in the *etc.tags.description* field when an overwritten dataset is loaded in the workspace. When clicking on the *Edit description* button the following dialog in Figure 5 below will appear.

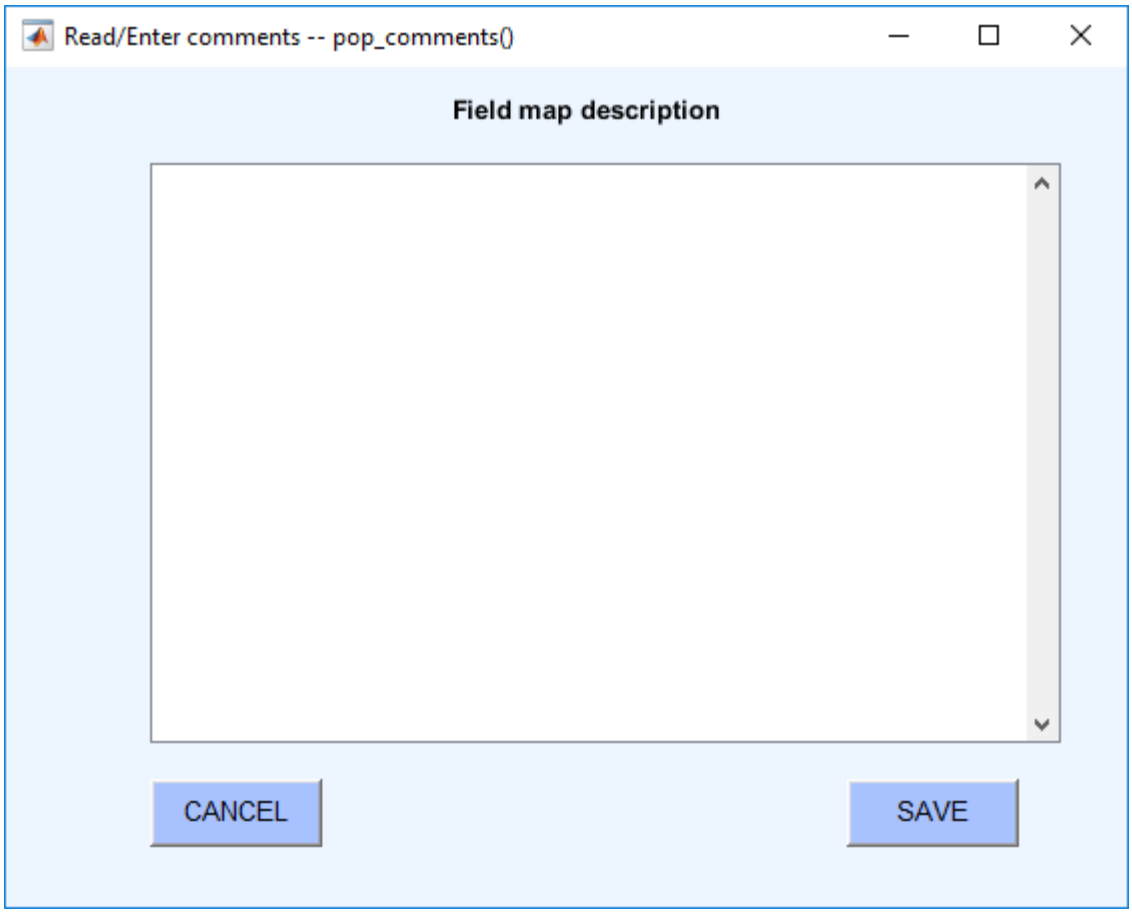

Figure 7. Modifying a field map description

The dialog above allows you to give the field map a description. The description will typically consist of details about the field map and why it exists. You can include what study the field map is used for along with the events that it contains. When done filling out the description, click the *Save* button.

The *pop_tageeg* function can be executed through the command-line. It takes in one argument which is an EEG dataset that is loaded into the workspace.

Example 2.1: Tag an *EEG* dataset with additional arguments using a menu.

```
[EEG, fMap, com] = pop_tageeg(EEG);
```

The *EEG* return parameter is the original dataset with the tags written to it. The *com* return parameter contains the command string that you would have typed to execute the *pop_tageeg* function without using the menu. You can save the *com* string for future use if you would like to tag another dataset using the same options without the menu.

The *tageeg* function is what you want to use when working from the command-line to run a script or using an existing *fieldMap* to tag other datasets. Most often there will be no user intervention when directly executing the function, similar to the example below.

Example 2.2: Tag another dataset without user intervention using a *fieldMap*.

```
[EEG1, fMap1] = pop_tageeg(EEG, 'BaseMap', fMap, 'UseGui', false);
```

The *BaseMap* argument is a *fieldMap* object or the full path to a *fieldMap* object that stores existing event tags. The *UseGui* argument controls whether or not to use CTagger for each field that is selected for tagging. To find a list of all the input arguments refer to Table 1. In Example 2.2, all user intervention is off. The idea here is that you tag one dataset and then call *pop_tageeg* to tag related datasets with no additional work.

MATLAB Syntax

```
[EEG, fMap, excluded] = pop_tageeg(EEG)
[EEG, fMap, excluded] = pop_tageeg(EEG, 'key1', 'value1', ...)
```

Table 1. A summary of arguments for pop_tageeg.

Name	Type	Description
`EEG`	Required	A structure containing data.
`'BaseMap'`	Name-Value	A fieldMap object or the name of a file that contains a fieldMap object to be used to initialize tag information.
`'ExcludeFields'`	Name-Value	A one-dimensional cell array of field names in the .event substructure to ignore during the tagging process. By default the following subfields of the .event structure are ignored: .latency, .epoch, .urevent, .hedtags, and .usertags. The user can over-ride these tags using this name-value parameter.
`'Fields'`	Name-Value	A one-dimensional cell array of fields to tag. If this parameter is non-empty, only these fields are tagged.
`'HedXML'`	Name-Value	Full path to a HED XML file. The default is the HED.xml file in the hed directory.
`'PreservePrefix'`	Name-Value	If false (default), tags for the same field value that share prefixes are combined and only the most specific is retained (e.g., /a/b/c and /a/b become just/a/b/c). If true, then all unique tags are retained.
`'PrimaryField'`	Name-Value	The name of the primary field. Only one field can be the primary field. A primary field requires a label, category, and a description tag. The default is the .type field.
`'SaveMapFile'`	Name-Value	A string representing the file name for saving the final, consolidated fieldMap object that results from the tagging process.
`'SelectFields'`	Name-Value	If true (default), the user is presented with a GUI that allow users to select which fields to tag.
`'UseGui'`	Name-Value	If true (default), the CTAGGER GUI is used to edit field tags.

2.2 Tagging a directory of datasets

To tag a directory of datasets click the *Tag directory* menu item under the *Tag files* menu under *File* which is illustrated below in Figure 5.

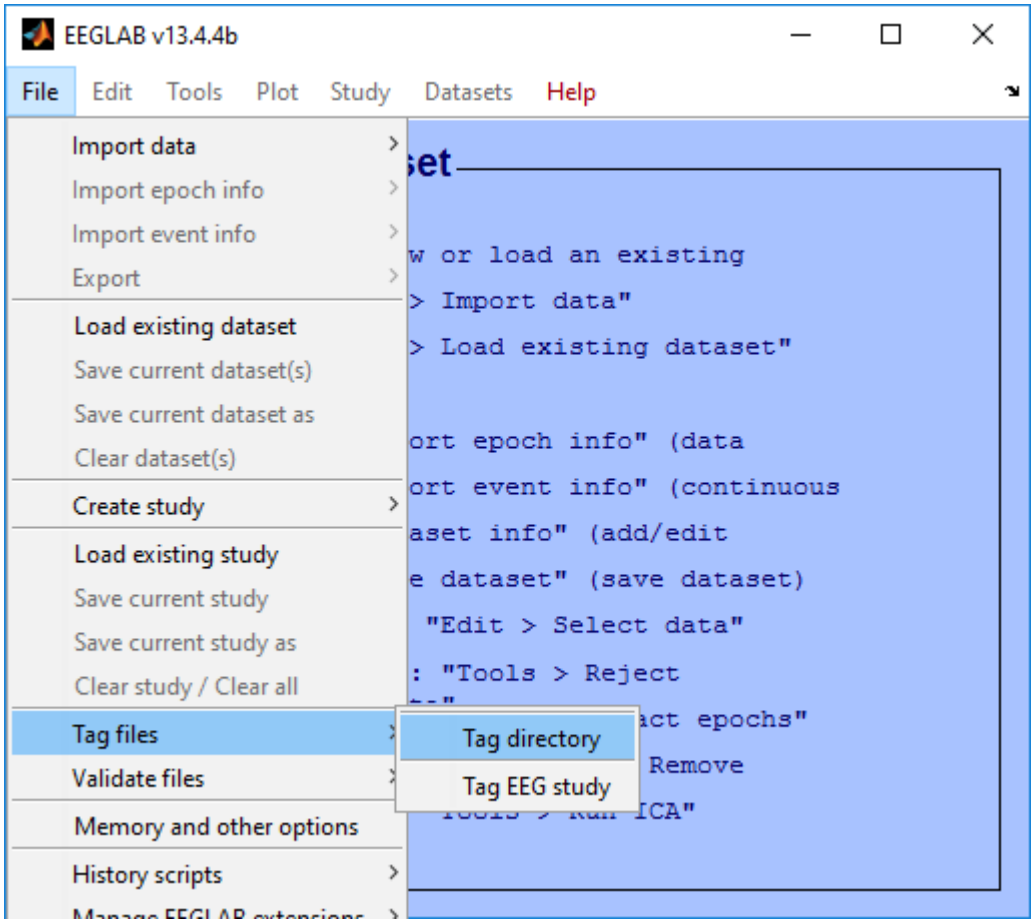

Figure 8. Tagging a directory of datasets from the EEGLAB File Menu

The *Tag directory* menu item executes the *pop_tagdir* function which brings up a menu for specifying options for tagging a directory of datasets. The function executes without reading any datasets into EEGLAB which is stored in memory.

The *pop_tagdir* function extracts tag information from all of the datasets stored in the directory tree and uses this information to list any existing tags. The function only considers *.set* datasets found in the directory.

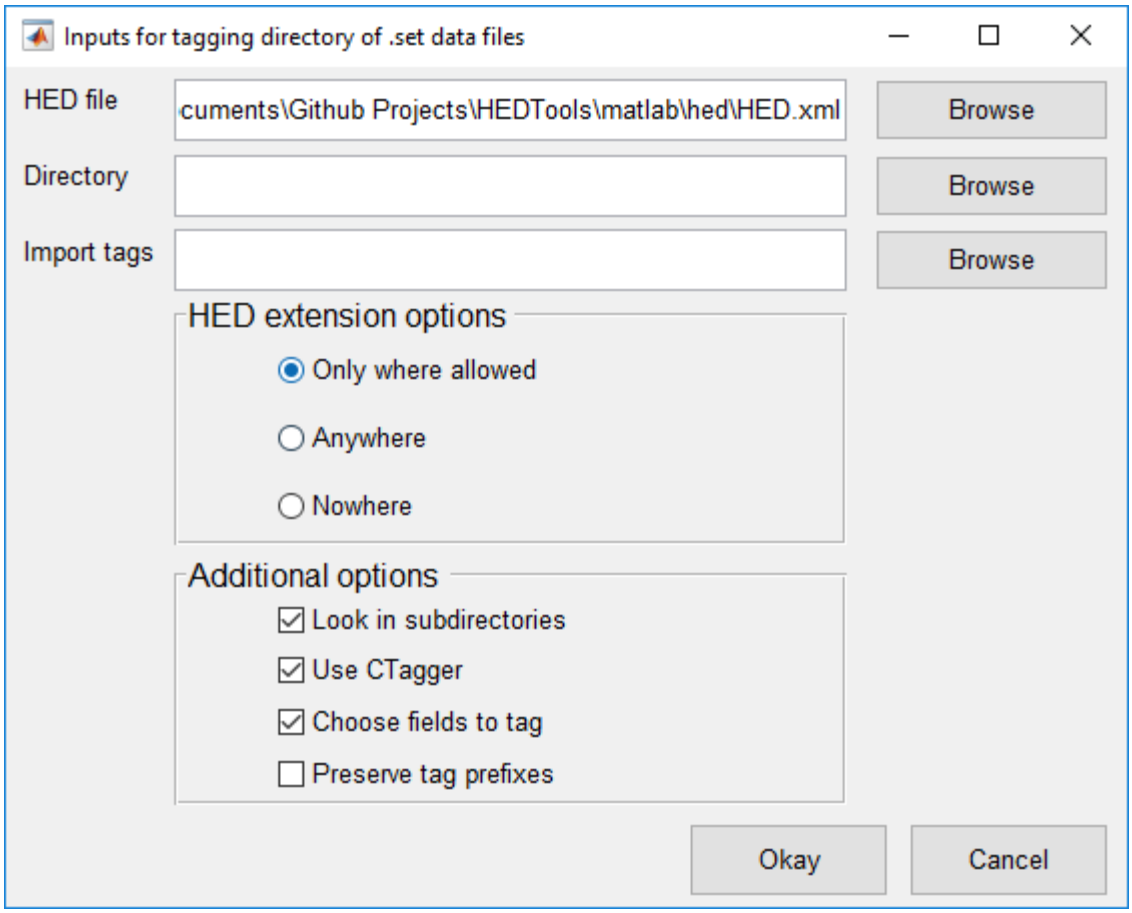

Figure 9. pop_tagdir menu

In Figure 9 above, the top section of the menu allows you to browse for a HED *.xml* file, a directory containing *.set* dataset files, and a *.mat* file containing tags to import. The HED file by default will be set to the *HED.xml* file found in the *hed* directory. The next section allows you to specify where the HED can be extended. These options include:

- Only where allowed - The HED can be extended for leaf tags and tags that have the *extensionAllowed* attribute.
- Anywhere - The HED can be extended for all tags.
- Nowhere – The HED cannot be extended at all even for leaf tags and tags that have the *extensionAllowed* attribute.

The last section allows you to select additional options. These options include:

- Search subdirectories for .set datasets (*Look in subdirectories)*
- Use CTagger to tag each selected field (*Use CTagger)*
- Choose fields to tag through a menu (*Choose fields to tag)*
- List the tags that have the most specific tag starting with that prefix or share the same prefix (*Preserve tag prefixes).*

Example 2.3: Tag a directory with additional arguments using a menu.

```
[fMap, fPaths, com] = pop_tagdir();
```

The *fMap* return argument is a *fieldMap* object that contains all of the tags associated with each unique field value. The *fPaths* return argument is a cell array containing the full path names of the datasets tagged during this call. You can save the *com* string for future use if you would like to tag another directory using the same options without the menu.

By default, *pop_tagdir* displays a menu, similar to the one of Figure 3, allowing you to decide which fields to tag or exclude. Once you have picked the fields to tag, the *pop_tagdir* function displays CTagger (see Figure 4) for each selected field. After tagging, *pop_tagdir* writes the tag information back to the datasets.

Similar to the *tageeg* function, the *tagdir* function is what you want to use when working from the command-line. The following example illustrates the simplest use of *tagdir* for tagging of a directory.

Example 2.4: Tag the data in the *inDir* directory.

```
[fMap, fPaths, com] = pop_tagdir(inDir);
```

Example 2.5: Tag a directory without user intervention using a *fieldMap*.

```
[fMap1, fPaths, com] = pop_tagdir(inDir, 'BaseMap', fMap, ...
                                  'UseGui', false);
```

MATLAB Syntax

```
[fMap, fPaths, com] = pop_tagdir(inDir)
[fMap, fPaths, com] = pop_tagdir(indDir, 'key1', 'value1', ...)
```

Table 2. A summary of arguments for pop_tagdir.

Name	Type	Description
`inDir`	Required	A directory that contains similar EEG .set files.
`'BaseMap'`	Name-Value	A fieldMap object or the name of a file that contains a fieldMap object to be used to initialize tag information.
`'DoSubDirs'`	Name-Value	If true (default), the entire inDir directory tree issearched. If false, only the inDir directory issearched.
`'ExcludeFields'`	Name-Value	A one-dimensional cell array of field names in the.event substructure to ignore during the taggingprocess. By default the following subfields of the.event structure are ignored: .latency, .epoch, .urevent, .hedtags, and .usertags. The user canover-ride these tags using this name-value parameter.
`'Fields'`	Name-Value	A one-dimensional cell array of fields to tag. If this parameter is non-empty, only these fields are tagged.
`'HedXML'`	Name-Value	Full path to a HED XML file. The default is the HED.xml file in the hed directory.
`'PreservePrefix'`	Name-Value	If false (default), tags for the same field value that share prefixes are combined and only the most specific is retained (e.g., /a/b/c and /a/b become just /a/b/c). If true, then all unique tags are retained.
`'PrimaryField'`	Name-Value	The name of the primary field. Only one field can be the primary field. A primary field requires a label, category, and a description. The default is the type field.
`'SaveDatasets'`	Name-Value	If true (default), save the tags to the underlying dataset files in the directory.
`'SaveMapFile'`	Name-Value	A string representing the file name for saving the final, consolidated fieldMap object that results from the tagging process.
`'SelectFields'`	Name-Value	If true (default), the user is presented with a GUI that allow users to select which fields to tag.
`'UseGui'`	Name-Value	If true (default), the CTAGGER GUI is displayed after initialization.

2.3 Tagging an EEGLAB study

To tag a directory of datasets click the *Tag study* menu item under the *Tag files* menu under *File* which is illustrated below in Figure 7.

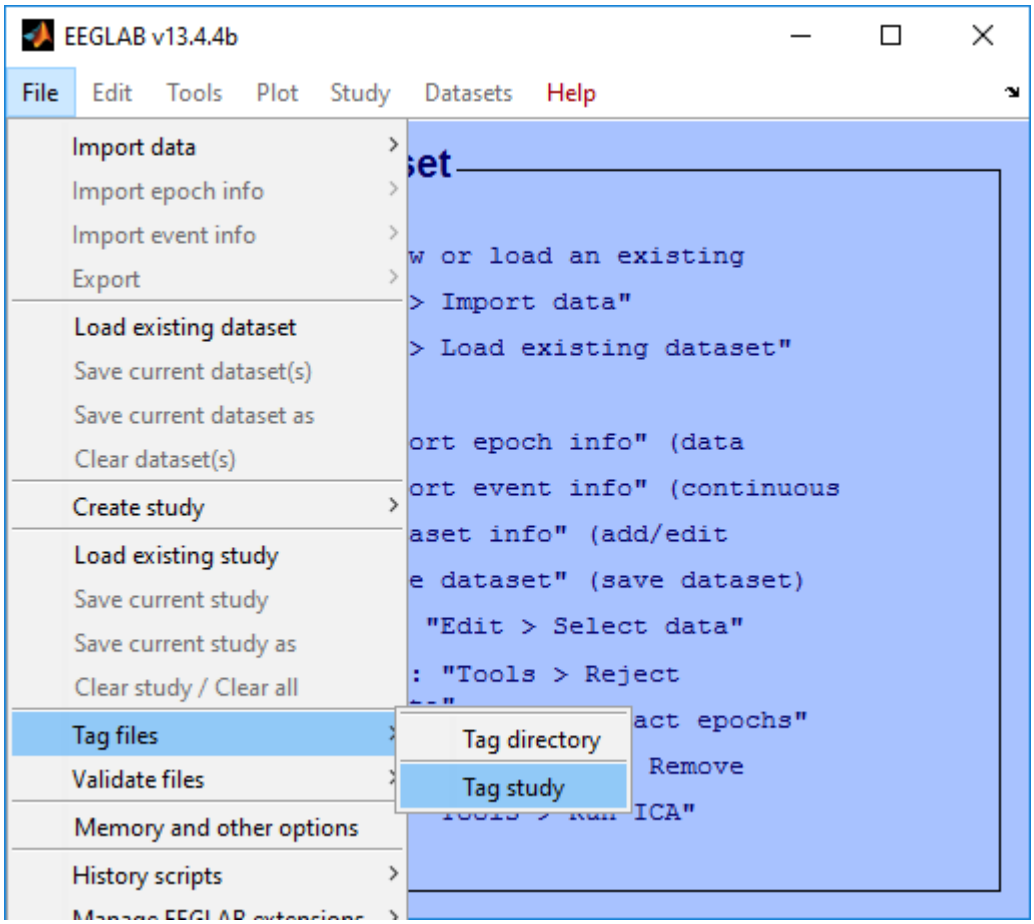

Figure 10. Tagging a study and its associated datasets from the EEGLAB File Menu

The *Tag study* menu item executes the *pop_tagstudy* function which brings up a menu for specifying options for tagging a study and its associated datasets. The function executes without reading any datasets into EEGLAB which is stored in memory.

The *pop_tagstudy* function extracts tag information from the study and uses this information to list any existing tags.

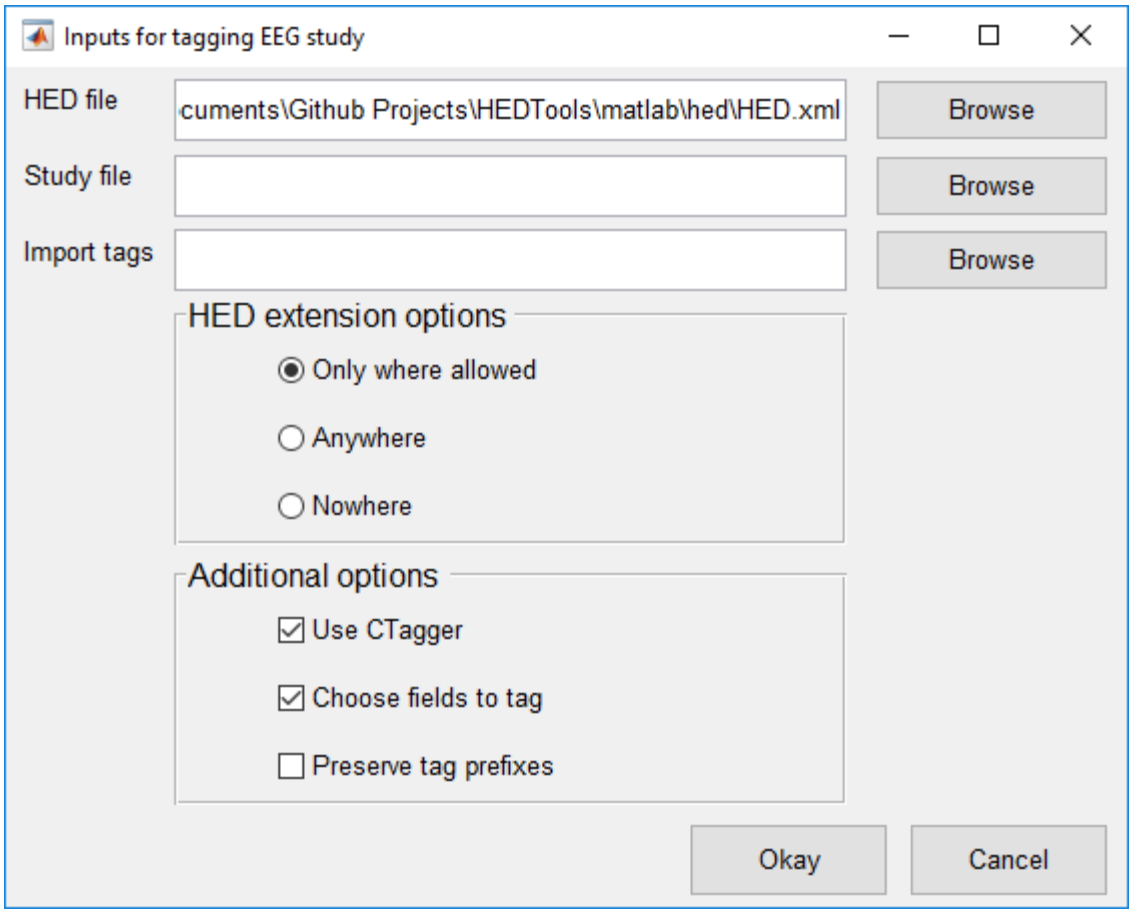

Figure 11. pop_tagstudy menu

In Figure 11 above, the top section of the menu allows you to browse for a HED *.xml* file, an EEGLAB *.study* file and a *.mat* file containing tags to import. The HED file by default will be set to the *HED.xml* file found in the *hed* directory. The next section allows you to specify where the HED can be extended. These options include:

- Only where allowed - The HED can be extended for leaf tags and tags that have the *extensionAllowed* attribute.
- Anywhere - The HED can be extended for all tags.
- Nowhere – The HED cannot be extended at all even for leaf tags and tags that have the *extensionAllowed* attribute.

The last section allows you to select additional options. These options include:

- Use CTagger to tag each selected field (*Use CTagger)*
- Choose fields to tag through a menu (*Choose fields to tag)*
- List the tags that have the most specific tag starting with that prefix or share the same prefix (*Preserve tag prefixes).*

Example 2.6: Tag a study with additional arguments using a menu.

```
[fMap, fPaths, com] = pop_tagstudy();
```

The *fMap* return argument is a *fieldMap* object that contains all of the tags associated with each unique field value. The *fPaths* return argument is a cell array containing the full path names of the datasets tagged during this call. You can save the *com* string for future use if you would like to tag another directory using the same options without the menu.

By default, *pop_tagstudy* displays a menu, similar to the one of Figure 3, allowing you to decide which fields to tag or exclude. Once you have picked the fields to tag, the *pop_study* function displays CTagger (see Figure 4) for each selected field. After tagging, *pop_tagstudy* writes the tag information back to the datasets.

Example 2.8: Tag a study without user intervention.

```
[fMap1, fPaths, excluded] = pop_tagstudy(studyFile, 'BaseMap', fMap,
'UseGui', false);
```

MATLAB Syntax

```
[fMap, fPaths, excluded] = pop_tagstudy(studyFile)
[fMap, fPaths, excluded] = pop_tagstudy(studyFile, 'key1', 'value1',
...)
```

Table 3. A summary of arguments for pop_tagstudy.

Name	Type	Description
`studyFile`	Required	The path to a EEG study.
`'BaseMap'`	Name-Value	A fieldMap object or the name of a file that contains a fieldMap object to be used for initial tag information.
`'ExcludeFields'`	Name-Value	A one-dimensional cell array of field names in the .event substructure to ignore during the tagging. By default the following subfields of the .event structure are ignored: .latency, .epoch, .urevent, .hedtags, and .usertags. The user canover-ride these tags using this name-value parameter.
`'Fields'`	Name-Value	A one-dimensional cell array of fields to tag. If this parameter is non-empty, only these fields are tagged.
`'PreservePrefix'`	Name-Value	If false (default), tags of the same event type that share prefixes are combined and only the most specific is retained (e.g., /a/b/c and /a/b become just /a/b/c). If true, then all unique tags are retained.
`'PrimaryField'`	Name-Value	The name of the primary field. Only one field can be the primary field. A primary field requires a label, category, and a description. The default is the type field.
`'SaveDatasets'`	Name-Value	If true (default), save the tags to the underlying files in the study.
`'SaveMapFile'`	Name-Value	The full path name of the file for saving the final, consolidated fieldMap object that results from the tagging process.
`'SelectFields'`	Name-Value	If true (default), the user is presented with a GUI that allow users to select which fields to tag.
`'UseGui'`	Name-Value	If true (default), the CTAGGER GUI is displayed after initialization.

3. Validating Data

There are three options for validating HED tags from the EEGLAB menus: *Validate current EEG* (from the *Edit* menu), *Validate study* (from the *Validate files* submenu under the *File* menu), or *Validate directory* (from the *Validate files* submenu under the *File* menu).

3.1 What the validation checks for

Aside from the validation checking if the event tags are present in the HED, the validation also checks for the following which will generate errors:

- Tags with the **isNumeric** attribute must be a numerical value. Some tags that are numerical have units associated with them that can be specified. If not, the default unit will be assigned to them

which is determined by its **unit class**. A **unit class** contains a collection of similar units. When a unit is specified for a numerical tag then its unit is checked to make sure that is a valid unit for that particular tag.

- Tags with the **required** attribute must be present in each and every event. These currently are tags that start with the prefixes Event/Category, Event/Description, and Event/Label.
- Tags with the **requireChild** attribute cannot be present in any event. Instead a descendant of these tags will have to be in its place. For example, the tag Event/Category cannot be present in an event. However, Event/Category/Participant response can because it is a descendant of Event/Category and doesn't have the **requireChild** attribute.
- Tags with the **unique** attribute can only have one descendant tag present in an event. For example, there cannot be two tags start with the prefix Event/Label because this tag has the **unique** attribute.
- Tags in groups can have no more than 2 tildes. For example, (Participant ~ Action/Type/Allow/Access ~ Item/Object/Person/ID Holder) is a valid group containing tildes.

In addition to this, the validation checks the syntax of the HED tags for the following which will generate warnings:

- The first word in each tag should be capitalized and all subsequent words should be lowercase. For example, /Event/Category/Experimental Stimulus is discouraged. The Stimulus part should be lowercase. Tags that take values do not have to have to meet this requirement.

Any event tags that do not comply with these rules will be written to a log file. The log file by default will only contain errors. To include warnings in the file you will need to specify the option. A typical log file will look like the following:

```
Issues in event 28:
      ERROR: Invalid HED tag - "Action/Type/Button press/Keyboard in group
      ((Participant ~ Action/Type/Button press/Keyboard ~ Participant/Effect/Body
      part/Arm/Hand/Finger))"
```

The snippet above contains the event that the issue occurred in, the tag that generated the issue, and the type of issue.

3.2 Validating a single dataset

First load a dataset into the workspace by clicking the *Load existing dataset* menu item under the *File* menu. To validate the dataset click the *Validate current dataset* menu item under the *Edit* menu which is illustrated below in Figure 9.

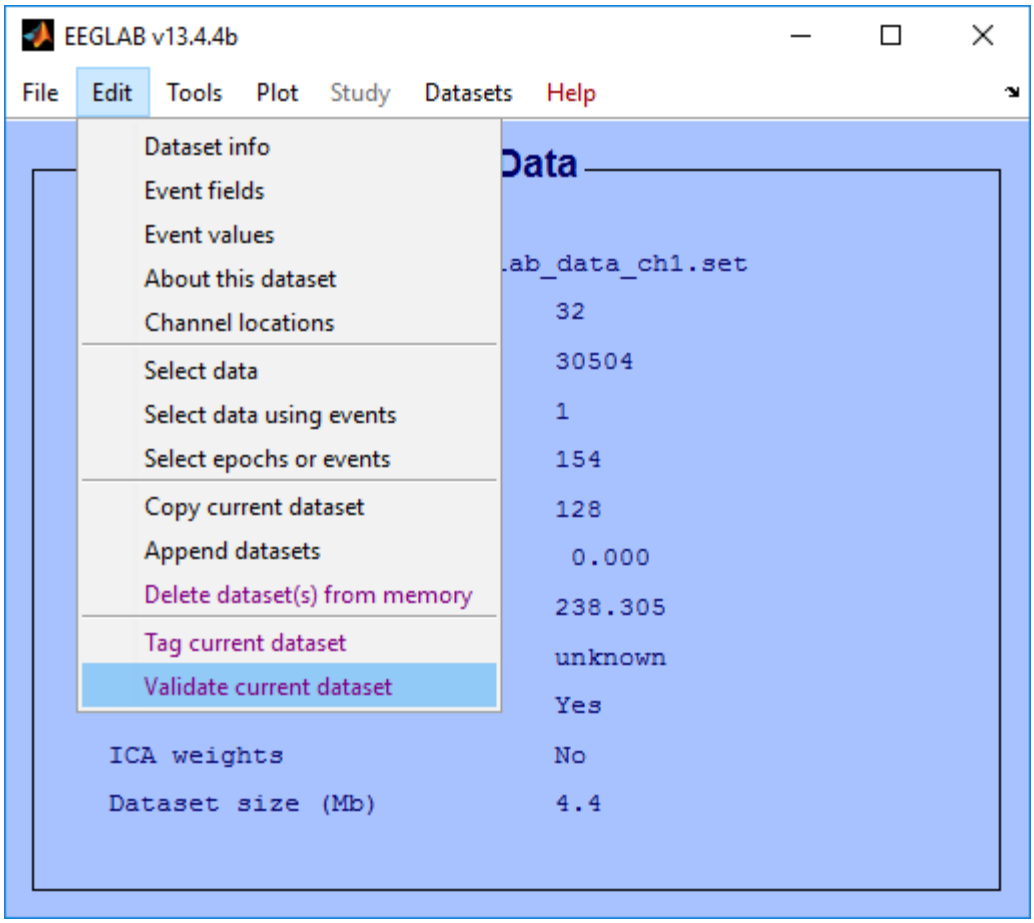

Figure 12. Validating the current dataset from the EEGLAB Edit Menu

The *Validate current dataset* menu item executes the *pop_validateeeg* function which brings up a menu for specifying options for validation.

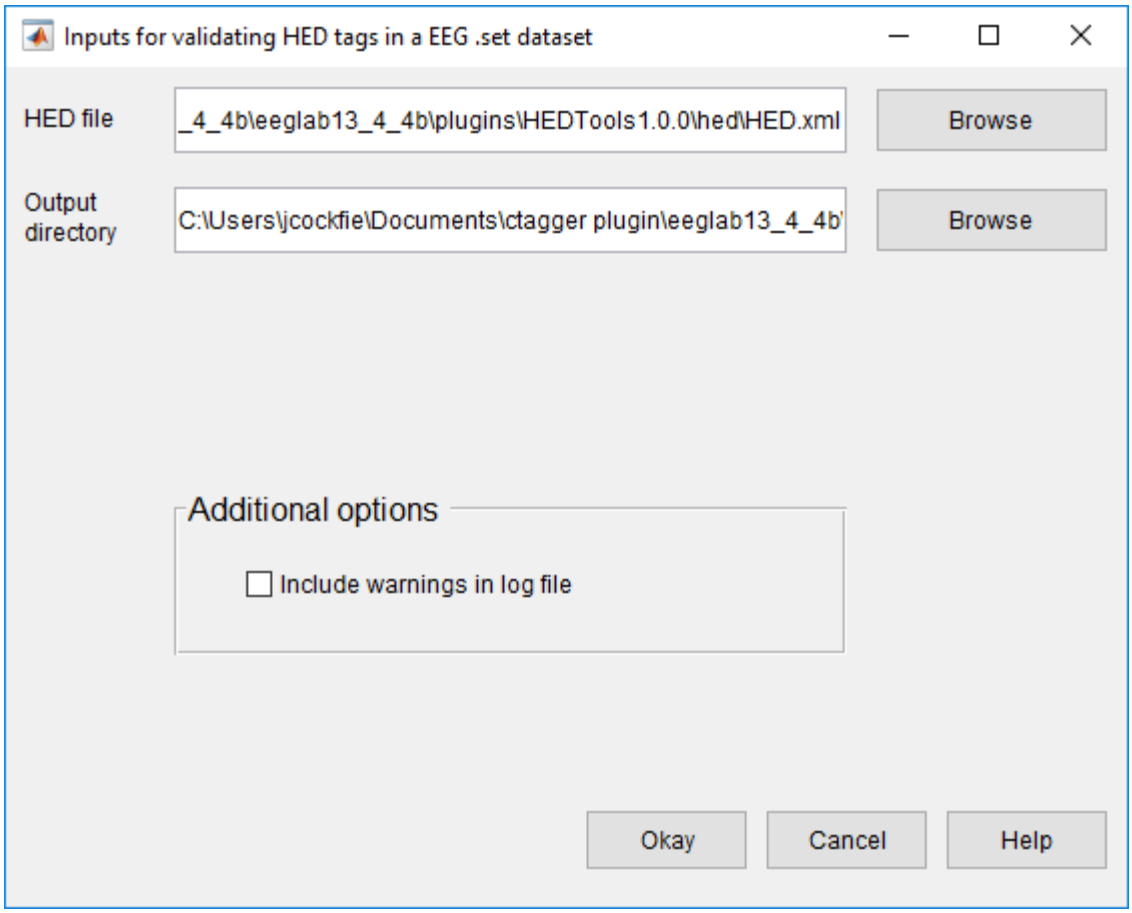

Figure 13. pop_validateeeg menu

In Figure 13 above, the top section allows you to browse for a HED file and an output directory. The next section allows you to select various options. Options include:

- Include warnings in addition to errors in the log file (*Include warnings in log file)*

Once all options are set click the *Okay* button to proceed.

Example 3.1: Validate a dataset with additional arguments using a menu.

```
[issues, com] = pop_validateeeg(EEG);
```

The *issues* return argument is a one-dimensional cell array containing the output from the validation. Each cell corresponds to a particular event that raised an issue. You can save the *com* string for future use if you would like to validate another directory using the same options without the menu.

The pop_*validateeeg* function can also be executed with additional options which are specified in Table 4 below.

Example 3.2: Validate a dataset and only write the output to the workspace.

```
issues = pop_validateeeg(EEG, 'writeOutput', false);
```

Table 4. A summary of arguments for pop_validateeeg.

Name	Type	Description
`EEG`	Required	The EEG dataset structure containing HED tags in the .event field.
`'tagField'`	Name-Value	The field in .event that contains the HED tags. The default field is .usertags.
`'hedXML'`	Name-Value	The full path to a HED XML file containing all of the tags. This by default will be the HED.xml file found in the hed directory.
`'outDir'`	Name-Value	The directory where the log file is written to. The default directory will be the current directory.
`'writeOutput'`	Name-Value	If true (default), write the validation issues to a log file in addition to the workspace. If false only write the issues to the workspace.

3.3 Validating a directory of datasets

To validate a directory of datasets click the *Validate directory* menu item under the *File* menu which is illustrated below in Figure 11.

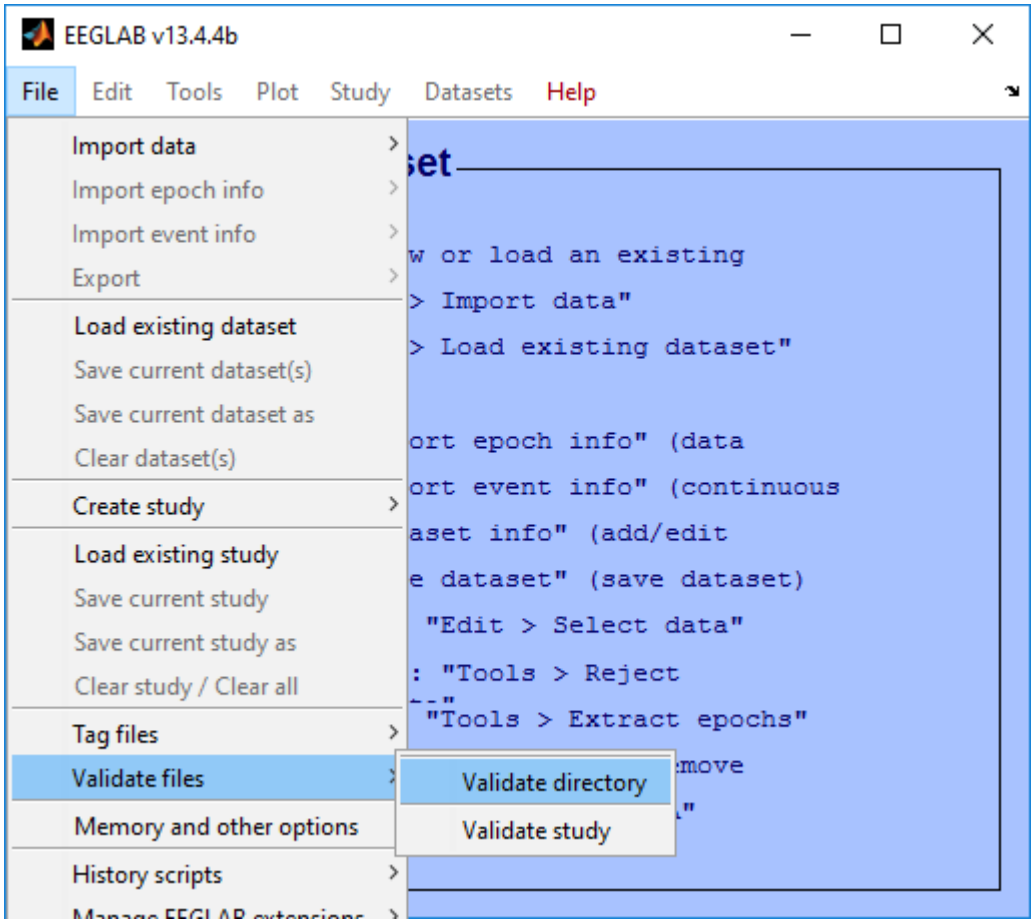

Figure 14. Validate a directory of datasets from the EEGLAB File Menu

The *Validate directory* menu item executes the *pop_validatedir* function which brings up a menu for specifying options for validation.

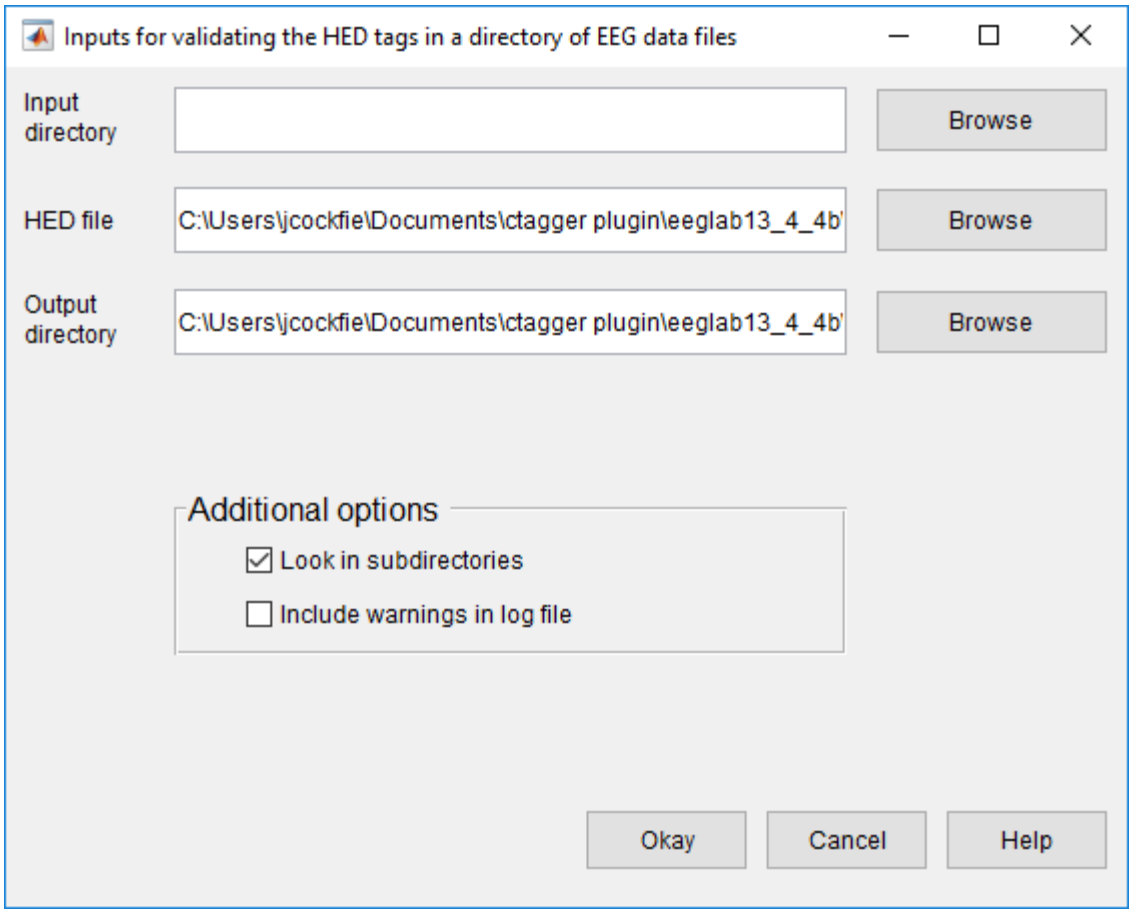

Figure 15. pop_validatedir menu

In Figure 15 above, the top section allows you to browse and select a root directory where the datasets are located, a HED file, and an output directory. The middle section allows you to select various options. Options include:

- Search in the subdirectories for datasets (*Look in subdirectories)*
- Include warnings in addition to errors in the log file (*Include warnings in log file)*

Once all options are set click the Okay button to proceed.

Example 3.3: Tag a study with additional arguments using a menu.

```
[fPaths, com] = pop_validatedir(inDir);
```

The *fPaths* return argument is a cell array containing the full path names of the datasets tagged during this call. You can save the *com* string for future use if you would like to validate another directory using the same options without the menu.

The pop_*validatedir* function can also be executed with additional options which are specified in the Table 5 below.

Example 3.4: Validate a directory of datasets and include warnings in the log files.

```
fPaths = pop_validatedir(inDir, 'generateWarnings', true);
```

Table 5. A summary of arguments for pop_validatedir.

Name	Type	Description
`inDir`	Required	A directory containing EEG datasets that will be validated.
`'doSubDirs'`	Name-Value	If true (default), the entire inDir directory tree is searched. If false, only the inDir directory is searched.
`'generateWarnings'`	Name-Value	If true, include warnings in the log file in addition to errors. If false (default), only errors are included in the log file.
`'tagField'`	Name-Value	The field in .event that contains the HED tags. The default field is .usertags.
`'hedXML'`	Name-Value	The full path to a HED XML file containing all of the tags. This by default will be the HED.xml file found in the hed directory.
`'outDir'`	Name-Value	The directory where the log files are written to. There will be a log file generated for each directory dataset validated. The default directory will be the current directory.

3.4 Validating an EEGLAB study

To validate a directory of datasets click the *Validate current dataset* menu item under the *File* menu which is illustrated below in Figure 16.

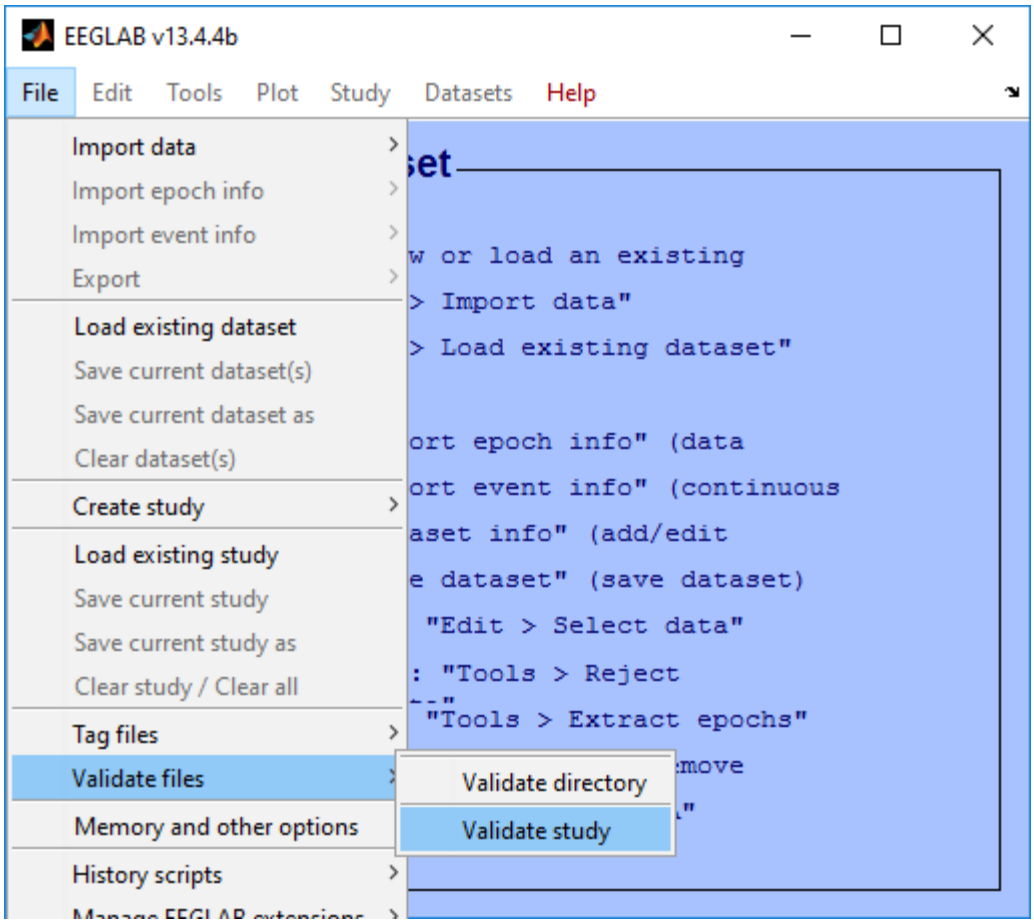

Figure 16. Validate a directory of datasets from the EEGLAB File Menu

The *Validate study* menu item executes the *pop_valideatestudy* function which brings up a menu for specifying options for validation.

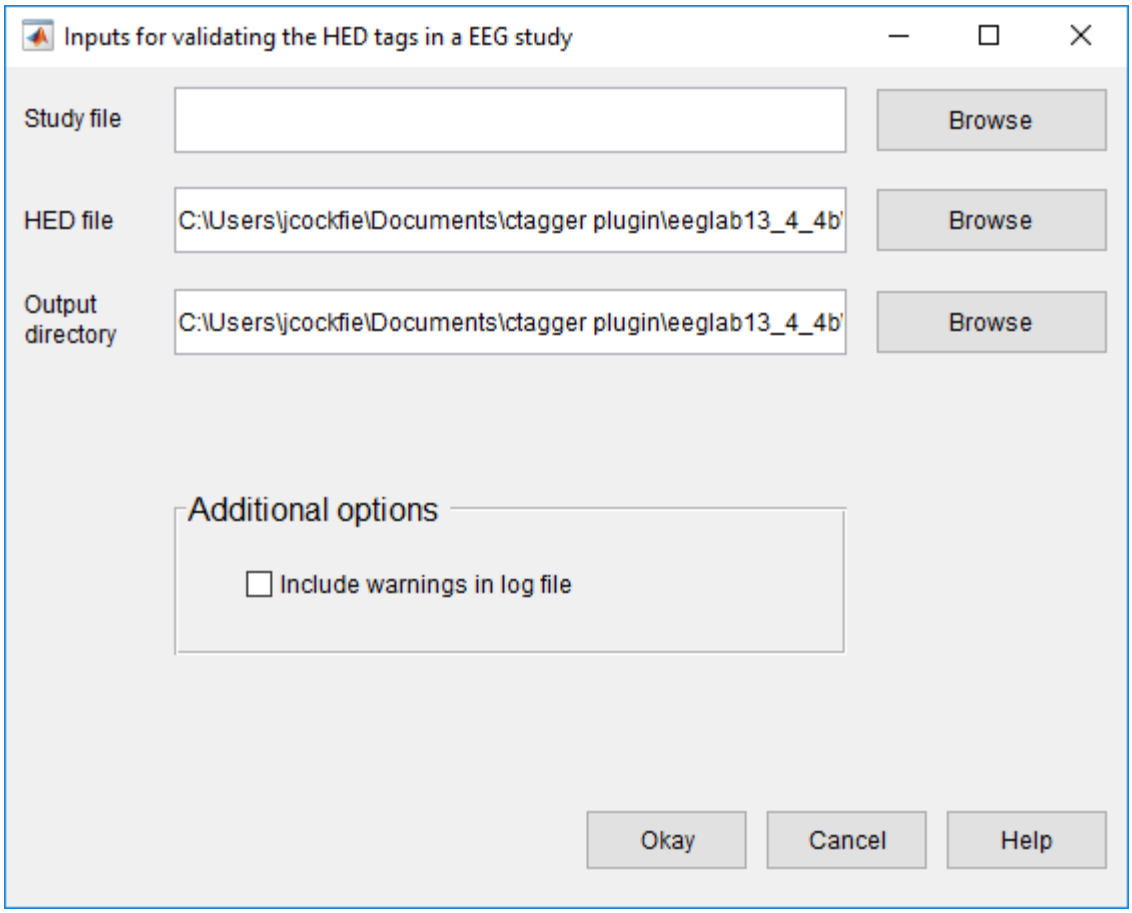

Figure 17. pop_validatestudy menu

In Figure 17 above, the top section allows you to browse and select a study file, a HED file, and an output directory. The middle section allows you to select various options. Options include:

- Include warnings in addition to errors in the log file (*Include warnings in log file)*

Once all options are set click the Okay button to proceed.

Example 3.5: Tag a study with additional arguments using a menu.

```
[fPaths, com] = pop_validatestudy();
```

The *fPaths* return argument is a cell array containing the full path names of the datasets tagged during this call. You can save the *com* string for future use if you would like to validate another directory using the same options without the menu.

The pop_*validatestudy* function can also be executed with additional options which are specified in Table 6 below.

Example 3.6: Validate a study and include warnings in the log files written to the current directory.

```
fPaths = pop_validatestudy(studyFile, 'generateWarnings', true);
```

Table 6. A summary of arguments for pop_validatestudy.

Name	Type	Description
`studyFile`	Required	The full path to an EEG study file.
`'generateWarnings'`	Name-Value	True to include warnings in the log file in addition to errors. If false (default) only errors are included in the log file.
`'hedXML'`	Name-Value	The full path to a HED XML file containing all of the tags. This by default will be the HED.xml file found in the hed directory.
`'tagField'`	Name-Value	The field in .event that contains the HED tags. The default field is .usertags.
`'outDir'`	Name-Value	The directory where the log files are written to. There will be a log file generated for each directory dataset validated. The default directory will be the current directory.

4. Extracting data epochs with HED tags

The EEGLAB *pop_epoch* function extracts data epochs time locked to specified event types. This function works well for datasets that have predefined event types (codes) corresponding to the different stimuluses and participant responses within the experiment. However, some datasets use latency in place of codes to identify the events. This is the case because there are so many different scenarios that can occur within a complex experiment that mirrors the real world. These kind of datasets are generally tagged to describe what transpires throughout each event.

The *pop_epochhed* menu shown in Figure 18 below, provides a simple way for extracting data epochs from annotated datasets. The idea here is that instead of extracting epochs based on event type, epochs are extracted based on the HED tags. The HED tags are assumed to be stored as a comma separated string in the *.usertags* field under the *.event* field of the EEG dataset. If the datasets that you are working with are not tagged, then please refer to the sections above in this manual on how to tag the datasets.

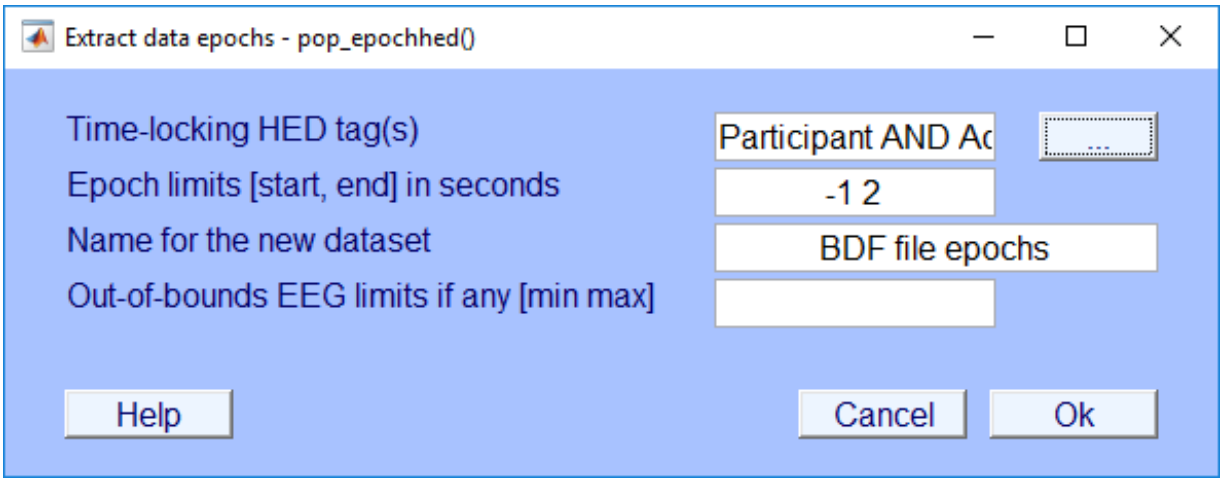

Figure 18. Input settings menu for epochhed

The *pop_epochhed* menu is almost identical to the EEGLAB *pop_epoch* menu with the exception of the first input field (*Time-locking HED tag(s)*). Instead of passing in or selecting from a group of unique event

types the user passes in a comma separated list of HED tags or a Boolean search string explained in the next section. Clicking the adjacent button (with the label …) will open a new menu used for inputting HED tags shown below in Figure 19.

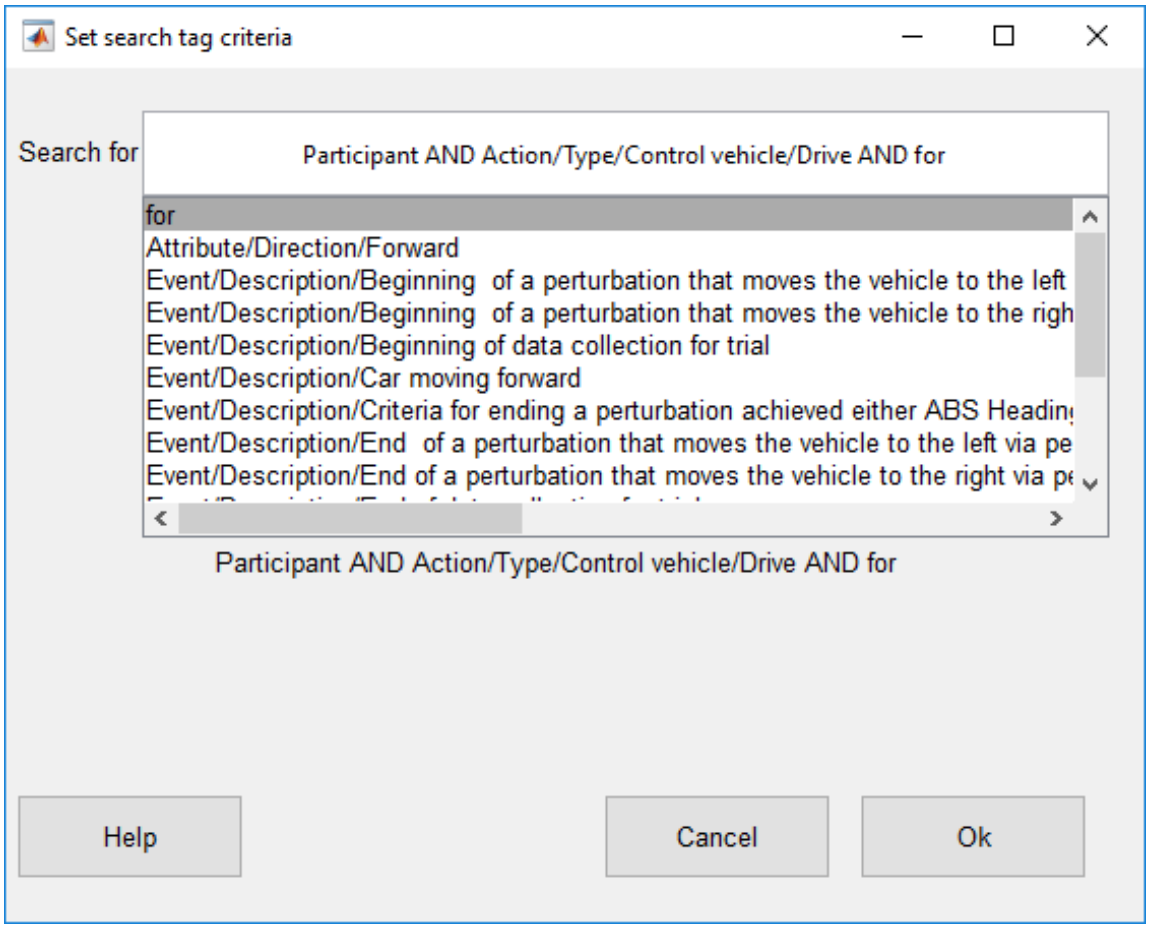

Figure 19. epochhed search bar

The advanced tag search uses Boolean operators (AND, OR, NOT) to widen or narrow the search. Two tags separated by a comma use the AND operator by default which will only return events that contain both of the tags. The OR operator looks for events that include either one or both tags being specified. The NOT operator looks for events that contain the first tag but not the second tag. To nest or organize the search statements use square brackets. Nesting changes the order of evaluation of the search statements. For example, "attribute/visual/color/green AND [item/2d shape/rectangle/square OR item/2d shape/ellipse/circle]" searches for events that have a green square or a green circle.

When you type something in the search bar it displays a list below containing possible matches. Pressing the "up" and "down" arrows on the keyboard while the cursor is in the search bar moves to the next or previous tag in the list. Pressing "Enter" selects the current tag in the list and adds the tag to the search bar. When done, click the "Ok" button to return to the main epoching menu.

5. Data Formats

HEDTools is structured and hence requires two items: a tag hierarchy and a map of field values. The tag hierarchy is in XML format and *HEDTools* provides a schema for validation. The map of tags are represented by a *fieldMap.*

5.1 XML tag hierarchy (HED)

HEDTools assume that rather than inventing tags at random, you will have a menu of suggested tags presented in hierarchical form as shown on the right in Figure 2. Internally, this hierarchy is represented as an XML string.

Example 5.1: A snippet from the HED 2.

```
<?xml version="1.0" encoding="utf-8"?>
<HED version="3.0.3">
   <node>
      <name>Event</name>
         <node position="1" predicateType="passThrough"
         requireChild="true" required="true">
            <name>Category</name>
            <description>This is meant to designate the reason
            this event was recorded</description>
             ...
```

The XML hierarchy shown in Example 5.1 is from *HED.xml* maintained specifically to support tagging of events in EEG experiments [1]. The Hierarchical Event Descriptor (HED) tags and supporting tools [2][3][4] [5] provide an infrastructure for data mining across data collections, once the datasets have been annotated.

HEDTools works with any XML file that conforms to the *HED.xsd*, the default XML schema specification. The *HED.xsd* schema is quite general, and you can substitute any XML hierarchy that conforms to the schema or build your own hierarchy from the ground up. You must take care in modifying the schema itself, as *HEDTools* assumes certain standard fields. The default XML hierarchy and validating schema are specified by the public constants *DefaultXml* and *DefaultSchema* in the *fieldMap* class defined in *helpers*.

5.2 Tags are path strings

Tags are simply path strings from the HED hierarchy. Each path string or tag uses forward slashes ("/") to separate the components in the path. Commas (",") separate multiple tags for the same event. **Do not use commas within text such as descriptions.** Users may group event tags with one level of parentheses to make the annotation clearer. Example 5.2 shows an example of the annotation for a stimulus event that consists of displaying a red circle in the center of the screen. The parentheses make it clear that the circle is red and located at the center of the screen. If the event designated the display of multiple objects of different colors, the parentheses would make the annotation more clear. The tagging also supports tag groups with embedded tilde ("~") characters to designate a sentence-like structure. **You can use only one level of parentheses that contains at most two tildes separating the subject from the predicate and the direct object.**

Example 5.2: Tag path string representation.

```
Event/Category/Experimental stimulus,
(Item/2D shape/Ellipse/Circle, Attribute/Visual/Color/Red,
Location/Screen/Center)
```

Normally, a tag that is more specific (i.e., the added tag has an existing tag as a prefix in string form or corresponds to an ancestor in the tag hierarchy) replaces a less specific tag. However, most *HEDTools* functions take an optional *PreservePrefix* argument, which is *false* by default. If you set this argument to *true*, *HEDTools* keeps both tags.

Example 5.3: When *PreservePrefix* argument is *true*, *HEDTools* keeps all versions of the tags.

```
Event/Category/Experimental stimulus
Event/Category/Experimental stimulus/Instruction/Attend
```

5.3 Field and tag map representations as a MATLAB structure

A field map (implemented by the MATLAB *fieldMap* class) associates field names with tag maps (implemented by the MATLAB *tagMap* class). A tag map associates tags with a group of values identified by a name (the "field"). The discussion of this section assumes type/subtype encoding (as illustrated in the next example) to simplify the discussion. However, field maps and tag maps do not rely on a specific representation.

Example 5.4: An experiment has two types of events, a stimulus and a user button press response, that are encoded as *STIM*, *RT*, respectively. The stimulus consists of a circle presented in one of three positions: to the left, center, or right of the screen. The positions are encoded by the researcher with numeric codes 1, 2, and 3 respectively. If the dataset is in EEGLAB format, an event such as a circle presented on the left side of the screen at 162 ms after the experiment begins might be stored as a structure:

```
EEG.event(1) =
               type: 'STIM'
               stimpos: 1
               latency: 162.048
               urevent: 1
```

Only the *.type* and *.stimpos* fields of the *.event* substructure are relevant for tagging. The *.urevent* is an EEGLAB-specific field that relates this event to the original event encodings, while *.latency* specifies the time of this event in frames.

HEDTools creates a *fieldMap* object to hold the tag map information for each of the two fields or groups: *type* and *stimpos*. The tag map for *type* contains the associations between each of its two values (*STIM* and *RT*) and the corresponding tags.

Example 5.5: The structure representation of the field map corresponding to Example 5.4 is:

```
fMap =
      xml: '<?xml version="1.0" ...'
      map: [1x2 struct]
```

Each of the *.map* structures corresponds to a tag map structure as shown in the next two examples.

Example 5.6: The structure representation of the tag map *stimpos* corresponding to the field map of Example 5.5:

```
fMap.map(1) =
      field: 'stimpos'
      values: [1x3 struct]

fMap.map(1).values(1) =
      code: '1'
      tags: {'Item/2D shape/Eclipse/Circle', 'Event/Description/Display
      of circle on left side of screen'}

fMap.map(1).values(2) =
      code: '2'
      tags: {'Item/2D shape/Eclipse/Circle','Event/Description/Display
      of circle in the center of the screen'}
```

```
fMap.map(1).values(3) =
      code: '3'
      tags: {'Item/2D shape/Eclipse/Circle','Event/Description/Display
      of circle on right side of screen'}
```

The *tag map* for the *type* field has the form:

```
fMap.map(2) =
      field: 'type'
      values: [1x2 struct]
```

In summary, a *field map* is a collection of *tag maps*, each identified by a group or field name. Field maps can be represented by a MATLAB structure that has two fields (*.xml*, and *.map*) at the top level. The *.xml* is a string representation of the tag hierarchy used for this tagging. Internally, *CTAGGER* represents a field map by a *fieldMap* object.

A tag map is an association of tags with a group of values identified by a name (field). *HEDTools* represents tag maps by a MATLAB structure that has two fields (*.field*, and *.values*) at the top level. The *.values* field contains a structure with two fields (*.code* and *.tags*).

5.4 How tags are stored in a dataset

Tags can be stored in any dataset that is a MATLAB structure. *HEDTools* assumes that the dataset itself is a structure and can store a representation of a field map in the *etc.tags* field of the dataset. One approach is to write the entire structure to the dataset.

Example 5.7: Storing the field map structure of Example 5.6 in the dataset *s* as a structure.

```
s.etc.tags = fMap;
```

It is also possible to store multiple maps by making *s.etc.tags* a structure array. For datasets that have events represented as a structure with fields, you can store the tags applicable to a particular event.

Example 5.8: The tag information stored in the individual event of Example 5.4.

```
EEG.event(1) =
      type: 'STIM'
      stimpos: 1
      latency: 162.048
      urevent: 1
      hedtags: ... direct mapped tags as a string
      usertags:'Item/2D shape/Eclipse/Circle'
```

The tags associated with a *type* value *STIM* as well as a *stimpos* value *1* are consolidated in *EEG.event(1).usertags* to allow data-mining. *CTAGGER* extracts these tags from a field map that is also maintained to allow revision and remapping. Tags from automated annotation at acquisition are stored in *.hedtags* and are not able to be remapped. The true tags for a particular event consist of the union of the tags in the *.hedtags* and *.usertags* fields.

5.5 The *fieldMap* object

The *fieldMap* class manages a collection of named groups and the mappings of their values to tags.

Example 5.9: Storing a collection of mappings in a *fieldMap* object.

```
f = fieldMap();
for k = 1:length(fMap.map)
   f.addValues(fMap.map(k).field, fMap.map(k).values, 'Merge');
end
```

The first statement creates an empty *fieldMap* object using the default XML. The loop adds the individual group mappings to the object. You can create multiple *fieldMap* objects and save them separately from the data. This allows you to maintain multiple tag mappings for different purposes.

MATLAB Syntax

```
fTags = fieldMap()
fTags = fieldMap('key1', 'value1', ...)
```

Table 7. A summary of arguments for fieldMap constructor.

Name	Type	Description
`'Description'`	Name-Value	Description of this object.
`'PreservePrefix'`	Name-Value	If *false* (default), *HEDTools* combines tags of the same field value that share prefixes and retains only the most specific (e.g., /a/b/c and /a/b become just /a/b/c). If *true*, then *HEDTools* retails all unique tags.
`'XML'`	Name-Value	A string containing the HED tag hierarchy used to create this object.

Table 8. A summary of the public methods of the fieldMap class.

Method	Description
`addValues`	Include values in this object based on update type.
`clone`	Create a copy of this object.
`getDescription`	Return the description of this object.
`getFields`	Return the fields of this object.
`getJson`	Return the JSON string version of this object.
`getJsonValues`	Return a JSON array of the JSON of the tag maps for this object.
`getMap`	Return the *tagMap* object associated with a specified field name.
`getMaps`	Return the tag maps for this object as a cell array of *tagMap* objects.
`getPreservePrefix`	Return the *PreservePrefix* flag.
`getStruct`	Return this object as a structure.
`getTags`	Return the tag string associated with value event of field.
`getValue`	Return the value structure corresponding to specified field and key.
`getValues`	Return the values for field as a cell array of structures.
`getXml`	Return a string containing the xml.
`merge`	Combine another *fieldMap* with this object based on update type.
`mergeXml`	Merge an XML string with this object's *HedXML* if valid.
`removeMap`	Remove the tag maps associated with specified field name.
`setDescription`	Set the description of this object.

Table 9. A summary of the public static methods of the fieldMap class.

Method	Description
`loadFieldMap`	Load a field map from a *.mat* file that contains a *fieldMap* object.
`saveFieldMap`	Save a field map to a *.mat* file.
`validateXml`	Validate an XML string given an XML schema (can throw exception).

5.6 The tagMap object

Internally, the *fieldMap* class uses the *tagMap* class to provide a common format for holding the tagging information for one group of values. This class has static methods for translating to and from the other formats and for merging tag maps.

Example 5.10: Representation of *fMap.map(1)* of Example 5.6 as a *tagMap* object.

```
t = tagMap('Field', 'stimpos');
for k = 1:length(fMap.map(1).values)
   t.addValues(fMap.map(1).values(k), 'Merge', false);
end
```

The first statement creates a *tagMap* object representing the tag-value mapping for the group of values called *stimpos*. The second statement adds the actual mapping of tags to values.

MATLAB Syntax

```
tMap = tagMap()
tMap = tagMap('key1', 'value1', ...)
```

Table 10. A summary of arguments for tagMap constructor.

Name	Type	Description
`'Field'`	Name-Value	String identifying the group this map is associated with.

Table 11. A summary of the public methods of the tagMap class.

Method	Description
`addValue`	Add the value (a structure) to this object based on update type.
`clone`	Create a copy of this object.
`getField`	Return the field name corresponding to this object.
`getJson`	Return a JSON string version of this object.
`getJsonValues`	Return a JSON string array with JSON for values of this object.
`getCodes`	Return the codes of keys associated with this object.
`getStruct`	Return this object as a structure.
`getValue`	Return the value structure corresponding to specified label or key.
`getValues`	Return the values as a cell array of structures.
`getValueStruct`	Return the values as an array of structures.
`merge`	Combine the *tagMap* object info with this one.

Table 12. A summary of the public static methods of the tagMap class.

Method	Description
`json2Values`	Return a structure corresponding to a specified JSON string.
`values2Json`	Return a JSON string representation of a value structure array.

5.7 The *tagList* object

Similar to how the *fieldMap* class uses the *tagMap* class, the *tagMap* class uses the *tagList* class. The *tagList* class represents each individual value and the associated tags in the *tagMap* group. This class also has static methods for translating to and from the other formats and for merging tag lists.

Example 5.11: Create a *tagList* representing a green square that belongs to *tagMap* group type.

```
tMap = tagMap('Field', 'type');
tList = tagList('square');
tList.add({'Attribute/Visual/Color/Green','Item/2D
shape/Rectangle/Square'});
tMap. addValue(tList);
```

MATLAB Syntax

```
tList = tagList(code)
```

Table 13. A summary of arguments for tagList constructor.

Name	Type	Description
code	Required	A unique event code value associated with tags.

Table 14. A summary of the public methods of the tagList class.

Method	Description
add	Add valid tag or tag group to this tagList.
addList	Add a list of tags or tag group to this tagList.
addString	Add a string of valid tags or tag groups to this tagList.
clone	Clone this *tagList* object by making a copy of the tag maps.
getCode	Returns the code associated with this tagList.
getCount	Returns the number of tags and tag groups in this tagList.
getJsonValues	Returns a JSON string version of this *tagList* object.
getKeys	Returns the tag map keys for this tagList.
getStruct	Returns this *tagList* as a structure array.
getTags	Returns a cell array with all of the tags and tag groups in this tagList.
intersect	Keep only the tag map keys that are in this *tagList* and in tagList newList.
isMember	Returns true if value is a valid tag or tag group in this *tagList*.
remove	Remove the tag or tag group in the tag map of this tagList corresponding to value.
removePrefixes	Remove the tags in this tagList that are prefixes in existing groups.
setCode	Returns the code associated with this tagList.
union	Adds the tags given in *tagList newList* to those of this tagList.

Table 15. A summary of the public static methods of the tagList class.

Method	Description
`deStringify`	Create a cell array representing a comma-separated string of tags.
`getCanonical`	Returns a sorted version of a valid tag or tag group.
`removeGroupDuplicates`	Removes duplicates from a tag group based on prefix.
`separateDuplicates`	Returns a list of tags without duplicates from cellstr tlist.
`splitTildesInGroup`	Splits the tildes in the cellstr tag group.
`stringify`	Create a string from a cell array of strings or cellstrs.
`stringifyElement`	Create string from cellstr or from string.
`tagList2Json`	Convert a *tagList* to a JSON string.
`validate`	Validate the input as a valid tag or tag group.
`validateTag`	Validate a tag string.
`validateTagGroup`	Validate a cellstr containing a tag group.

6. Status and availability

The base *HEDTools* is currently available and undergoing user testing.

7. Acknowledgments

The authors acknowledge helpful conversations with Christian Kothe, Nima Bigdely Shamlo, Alejandro Ojeda, Arno Delorme, and Scott Makeig, all of University of California San Diego as well as Scott Kerick, Jeanne Vettel of the Army Research Laboratories, Tony Johnson, Michael Dunkel, and Michael Nonte of DCS Corporation, and Rob Geary and Andrew Moseley-Gholl of the University of Michigan. This research was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-10-2-0022. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory of the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

8. References

[1] N. Bidely-Shamlo, K. Kreutz-Delgado, M. Miyakoshi, M. Westerfield, T. Bel-Bahar, C. Kothe, J. Hsi, S. Makeig, and K. Robbins, "Hierarchical Event Descriptor (HED) Tags for Analysis of Event-Related EEG Studies," presented at the IEEE GlobalSIP, Austin, TX (submitted), 2013.

[2] "ESS - SCCN." [Online]. Available: http://sccn.ucsd.edu/wiki/ESS. [Accessed: 19-May-2013].

[3] "Simulation and Neuroscience Application Platform (SNAP)." [Online]. Available: https://github.com/sccn/SNAP. [Accessed: 08-Jun-2013].

[4] "XDF (Extensible Data Format)." [Online]. Available: https://code.google.com/p/xdf/. [Accessed: 08-Jun-2013].

[5] "HED - SCCN." [Online]. Available: http://sccn.ucsd.edu/wiki/HED. [Accessed: 11-Jun-2013].