HEDTools
User Manual

September 16, 2016

Table of Contents

Table of Figures

Table of Tables

1. Getting Started with CTAGGER Tools

1.1 Overview

HEDTools is an MATLAB/Java Toolbox and 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 meet the requirements of HED 3.

1.2 Requirements

HEDTools is dependent on MATLAB. If you are using *HEDTools* as a plugin then it will also be dependent on EEGLAB. Please use the most current version of EEGLAB. As mentioned in section 1.3.3 below, there is a separate user manual dedicated for the plugin version in which you should use.

1.3 Installation

You can run *HEDTools* as a standalone toolbox or as a plugin for EEGLAB.

1.3.1 Running with data files that only are .mat files

If your data files are *.mat* files, you can simply unzip the *HEDTools* anywhere you choose and set the current directory to the *matlab* directory. Execute the *setup* script to set the paths each time you run MATLAB. Alternatively, you can add the code contained in *setup* to your *startup* script. If you are not using EEGLAB, you can comment out the last section of the *setup* script.

1.3.2 Running with .set data file types

If you wish to use EEGLAB, you should follow the directions above without commenting out the last section of the `setup` script.

1.3.3 Running as a plugin to EEGLAB

To use the *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. You will not need to use any of the code from the *matlab* directory. There is a completely separate user manual that is dedicated to using the *HEDTools* as a plugin which can be found in the *documentation* directory under *HEDTools1.0.0.* Please refer to that one before proceeding.

1.4 Community tagging database

In order for *HEDTools* to behave as a community tagging system, it must be run with the back-end database (*PostrgreSQL)* that stores a common tag hierarchy. **However, the database is currently under development and will not be available for this distribution of *HEDTools*.**

2. Tagging Data

2.1 Tagging a single dataset

The *tageeg* function allows you to tag a single dataset either from a script, the command line with a menu, or from EEGLAB. When run from EEGLAB, you can tag the current dataset through the EEGLAB *Edit* menu. The dataset only needs to be a structure with an *.event* subfield and does not have to conform fully to EEGLAB requirements.

The following example illustrates the simplest use of *tageeg* for interactive tagging of an *EEG* structure. The input dataset structure can be any structure, but *HEDTools* extracts field and tag information only from the *EEG.etc.tags*, the *EEG.event*, and the *EEG.urevent* fields. If the data does not have any of these fields, *HEDTools* does not extract any information.

Example 2.1: Tag the data in the *EEG* structure.

```
[EEG, fMap] = tageeg(EEG)
```

The *tageeg* function extracts any tag information in the *EEG* structure and uses this information to list any existing tags. The user can then tag or update event tags through a tagging application (CTagger) that appears. When you press the proceed button on the CTagger it returns the *EEG* structure with new tags added in the *EEG.event.usertags* field of the event structure.

HEDTools also adds information in the *EEG.etc.tags* field, which contains a string containing the XML vocabulary used to tag the data as well a *fieldMap* object associating event field names with tags. The latter is used to enable easy editing and modification of the tagging in subsequent sessions. The *fMap* return argument is the *fieldMap* object that contains the tag map information created during this call. Users can pass this map as an optional argument to subsequent calls to *tageeg* to apply tagging created from one dataset to another dataset. Most researchers use the same event codes or labels for all of the datasets in a collection, and this enables them to reuse the tagging performed on one dataset for many others.

HEDTools allows the tagging of different fields in the *EEG.event* structure independently. Normally, you would just tag the unique values that appear in *EEG.event.type* field. However, if you added additional fields to *EEG.event* to better distinguish subcategories, the events are defined by the unique combinations of labels in multiple fields. You can select the fields to tag from a menu. The menu contains two lists which consists of fields to exclude and fields to include for tagging. This method requires that your labeling scheme be orthogonal --- that is, it assumes each field can be tagged separately and then the tags from the two fields combined to tag an event.

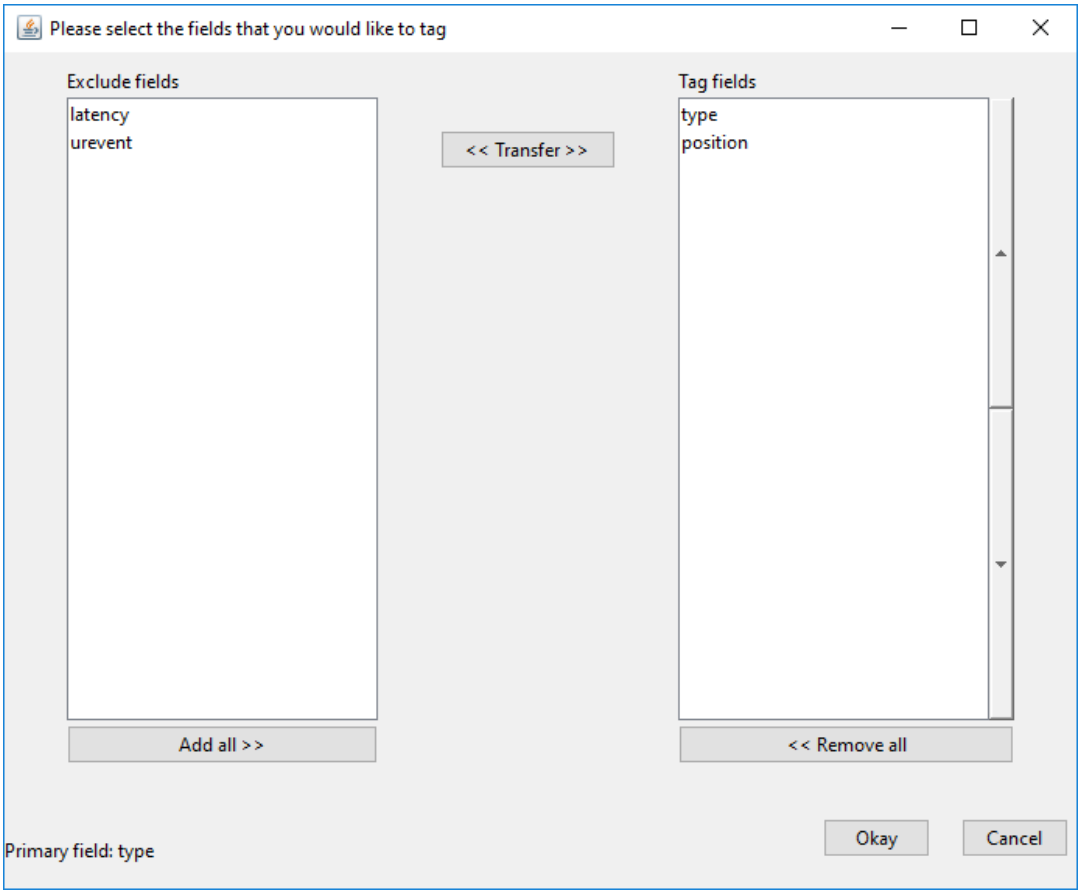

Figure 1. Field selection menu for choosing which fields to tag.

Figure 1 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.

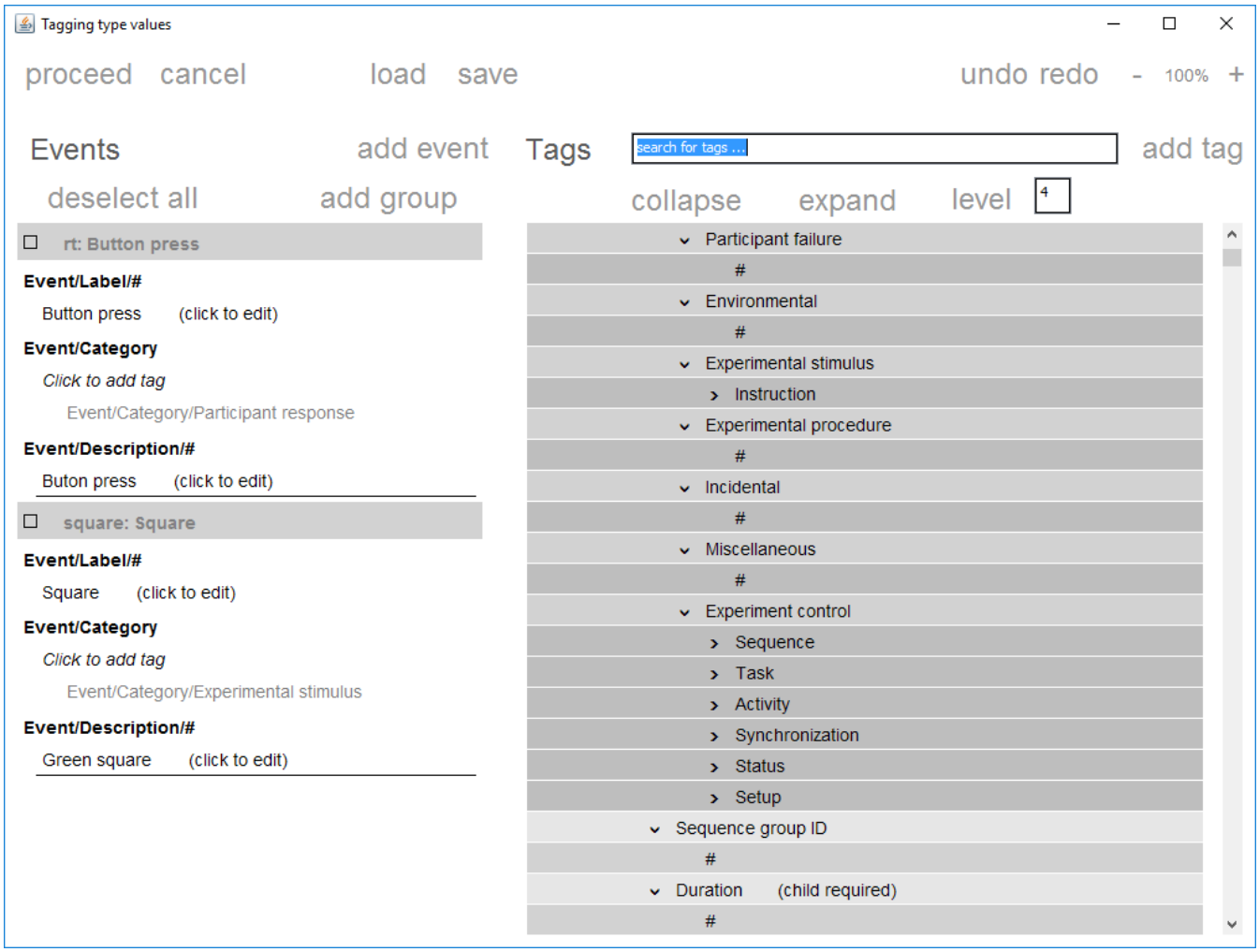

Figure 2. CTagger for the .type field.

In Figure 2, 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.

The *tageeg* function has several optional additional arguments specified in name-value pair format. You can access *tageeg* through a menu by calling *pop_tageeg*.

Example 2.2: Tag a dataset with additional arguments using a menu.

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

The *com* return parameter contains the command string that you would have typed to execute *tageeg* 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 using the CTagger.

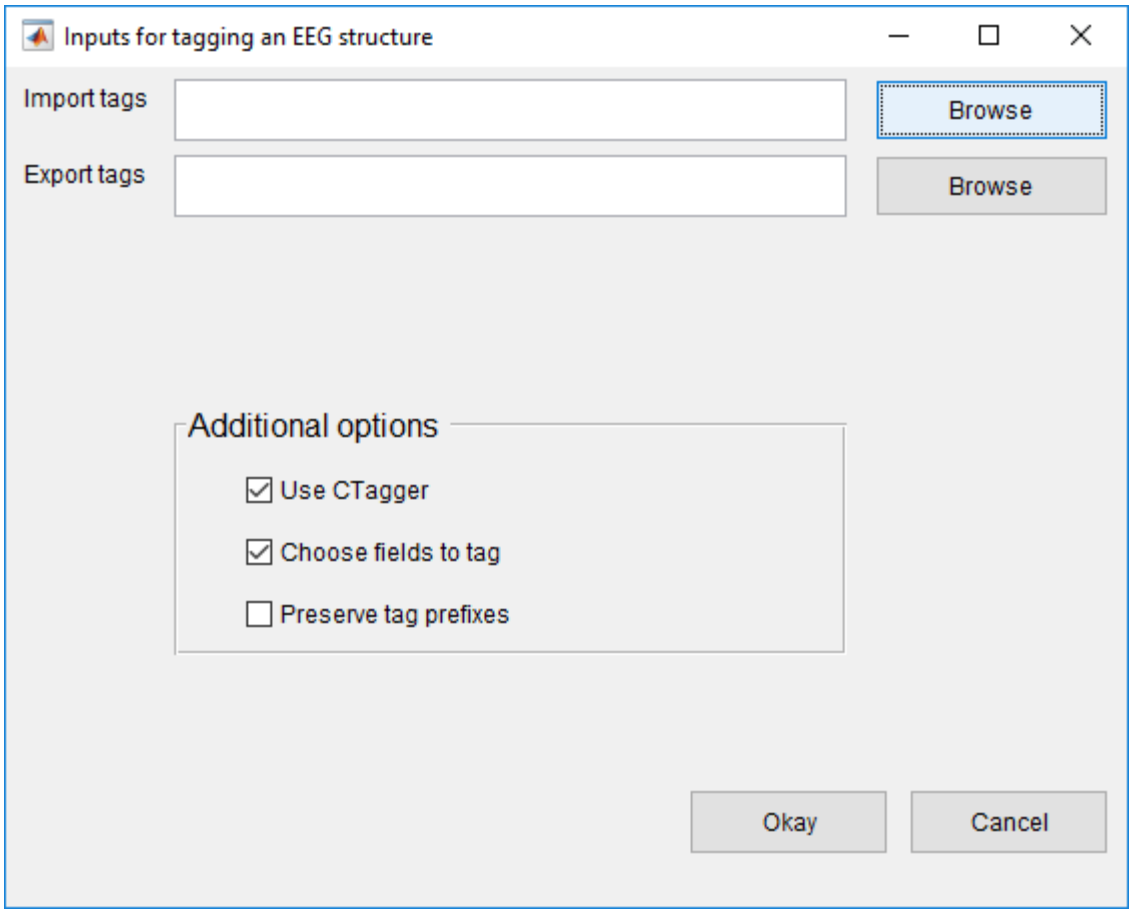

Figure 3. Input settings menu for tageeg.

In Figure 3 above, the top section of the menu allows you to browse and select files for importing and exporting event tags respectively. Import and export tag files are stored as *.mat* files. When browsing for an import file only *.mat* files will be looked for. The next 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.

Example 2.3: Tag another dataset without user intervention using a *fieldMap*.

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

The *UseGui* argument controls whether or not to use CTagger for each selected field. If this argument is set to false, then the menu used to select which fields to tag will not appear, even if *SelectFields* is set to true. In Example 2.3, all user intervention is off. The idea here is that you tag one dataset and then call *tageeg* to tag related datasets with no additional work.

MATLAB Syntax

```
[EEG, fMap, excluded] = tageeg(EEG)
[EEG, fMap, excluded] = tageeg(EEG, varargin)
```

Table 1. A summary of arguments for 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.
`'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

The *tagdir* function and its supporting functions (*tagdir_input* and *pop_tagdir*) allow you to tag an entire directory from a script with no user intervention, from the command line with a menu, or from EEGLAB. When run from EEGLAB, you can tag a directory through the EEGLAB *Tag files* submenu under the *File* menu without reading any datasets into EEGLAB.

The following example illustrates the simplest use of *pop_tagdir* for interactive tagging of a directory.

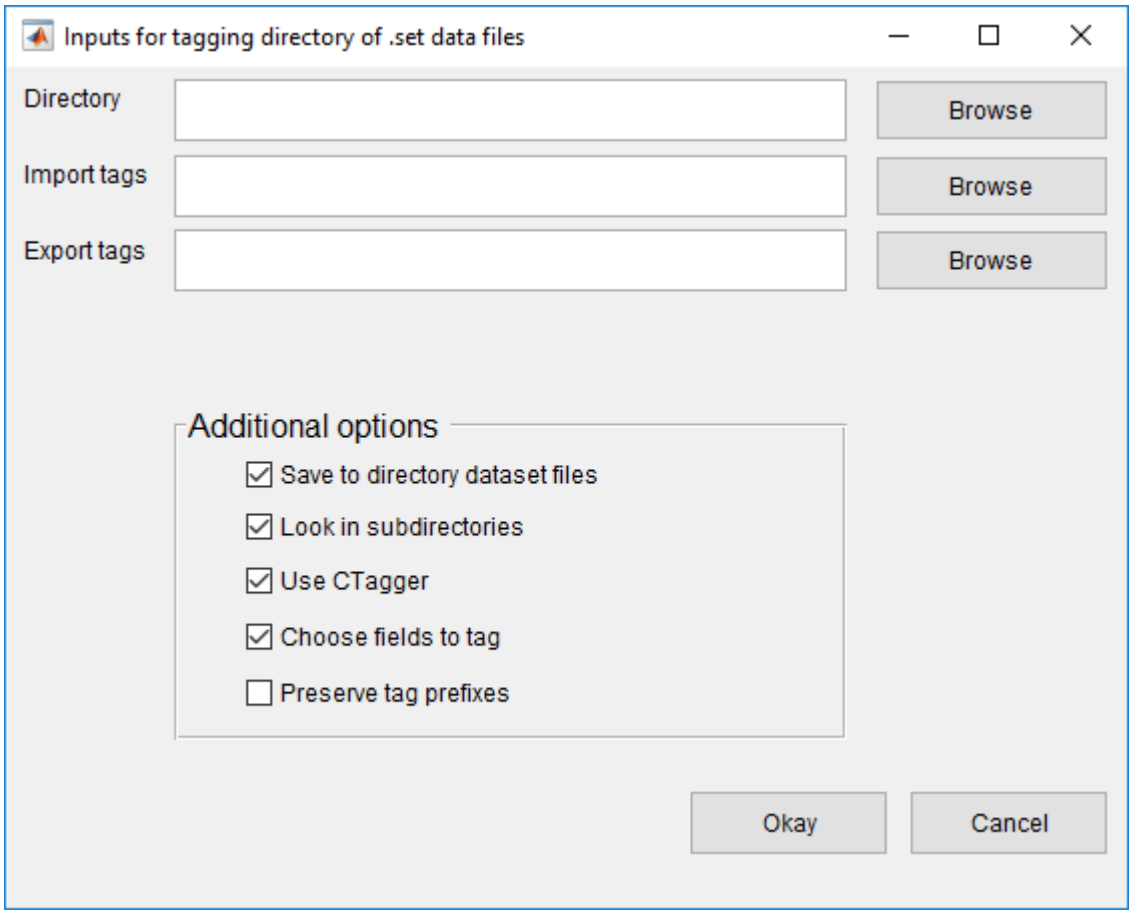

Figure 4. pop_tagdir menu.

In Figure 4 above, the top section of the menu allows you to browse and select a root a directory where the datasets are located and select files for importing and exporting event tags respectively. Import and export tag files are stored as *.mat* files. When browsing for an import file only *.mat* files will be looked for. The left middle section designates how to save the underlying *EEG* dataset files (if *Save to directory dataset files* is checked). The right middle section allows you to select various options. Options include:

- Write the tagged results to the datasets (*Save to directory dataset files)*
- 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 2.4: 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.

Example 2.5: Tag the data in the *inDir* directory.

```
[fMap, fPaths, excluded] = tagdir(inDir);
```

The *excluded* argument is a cell array of field names of excluded fields. *HEDTools* only considers *.set* datasets. By default, tagdir displays a menu, similar to the one of Figure 1, allowing you to decide which fields to tag or exclude. Once you have picked the fields to tag, the *tagdir* function displays the CTagger for each selected field. After tagging, *tagdir* writes the tag information back to the datasets.

You can also use the *tagdir* function with additional arguments similar to the *tageeg* function. Call the *pop_tagdir* function to use a menu to set these arguments.

Example 2.6: Tag a directory without user intervention using the tag map information of Example 2.5.

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

The *UseGui* controls whether or not you want to use CTagger for each selected field. In Example 2.6, all user intervention is off. The idea here is that you tag an entire directory once you have created your tag mapping.

MATLAB Syntax

```
[fMap, fPaths, excluded] = tagdir(inDir)
[fMap, fPaths, excluded] = tagdir(indDir, varargin)
```

Table 2. A summary of arguments for 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.
`'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

The *tagstudy* function and its supporting functions (*tagstudy_input* and *pop_tagstudy*) allow you to tag an EEGLAB study from a script, from the command line with a menu, or from EEGLAB. When run from EEGLAB, you can tag a study through the EEGLAB *File* menu.

The following example illustrates the simplest use of *pop_tagstudy* for interactive tagging of an EEGLAB study.

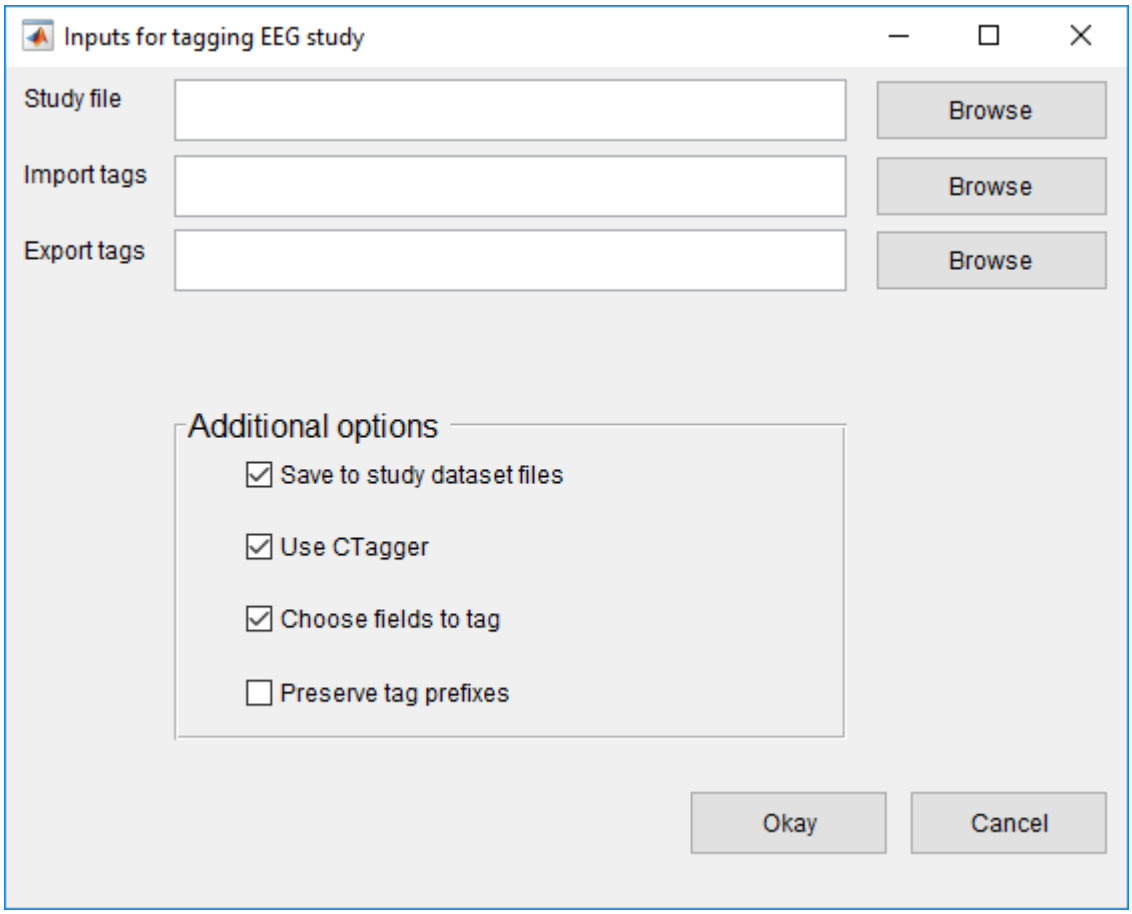

Figure 5. pop_tagstudy menu.

In Figure 5 above, the top section of the menu allows you to browse and select a study and select files for importing and exporting event tags respectively. Import and export tag files are stored as *.mat* files. When browsing for an import file only *.mat* files will be looked for. The left middle section designates how to save the underlying *EEG* dataset files (if *Save to directory dataset files* is checked). The right middle section allows you to select various options. Options include:

- Write the tagged results to the datasets (*Save to study dataset files)*
- 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 2.7: 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.

Example 2.8: Tag the data represented by the EEGLAB study specified by the *studyFile* file.

```
[fMap, fPaths, excluded] = tagstudy(studyFile);
```

The *excluded* return argument is a cell array of field names of excluded fields. When you call *tagstudy*, *HEDTools* presents menu, similar to the one of Figure 1. This menu allows you to decide which fields to tag or exclude. Once you have picked the fields to tag, the *tagstudy* function displays CTagger for each selected field. After you have completed your tagging, *tagstudy* writes the tag information back to the datasets and the study file, depending on the save options that you have selected.

You can also call the *tagstudy* function with additional arguments.

Example 2.9: Tag a study without user intervention using tag map information of Example 2.8.

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

The *UseGui* controls whether or not you want to use CTagger for each selected field. In Example 2.9, all user intervention is off. The idea here is that you tag a study without user intervention once you have created your tag mapping.

MATLAB Syntax

```
[fMap, fPaths, excluded] = tagstudy(studyFile)
[fMap, fPaths, excluded] = tagstudy(studyFile, varargin)
```

Table 3. A summary of arguments for 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:

- Tags shouldn't begin or end with a slash. For example, /Event/Category/Experimental stimulus and Event/Category/Experimental stimulus/ are discouraged.
- 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.

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:
    "Action/Type/Button press/Keyboard in group ((Participant ~ Action/Type/Button
press/Keyboard ~ Participant/Effect/Body part/Arm/Hand/Finger))" is not a valid HED
tag
```

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

The *validateeeg* function and its supporting functions (*validateeeg_input* and *pop_validateeeg*) allows you to validate a single dataset either from a script, the command line with a menu, or from EEGLAB. When run from EEGLAB, you can validate the current dataset through the EEGLAB *Edit* menu. The dataset only needs to be a structure with an *.event* subfield and does not have to conform fully to EEGLAB requirements.

The following example illustrates the simplest use of *validateeeg* for interactive validating of an *EEG* structure. The *HEDTools* validates the tags that are found in *EEG.event*. If there are no tags in the dataset please refer to section 2 for tagging the dataset.

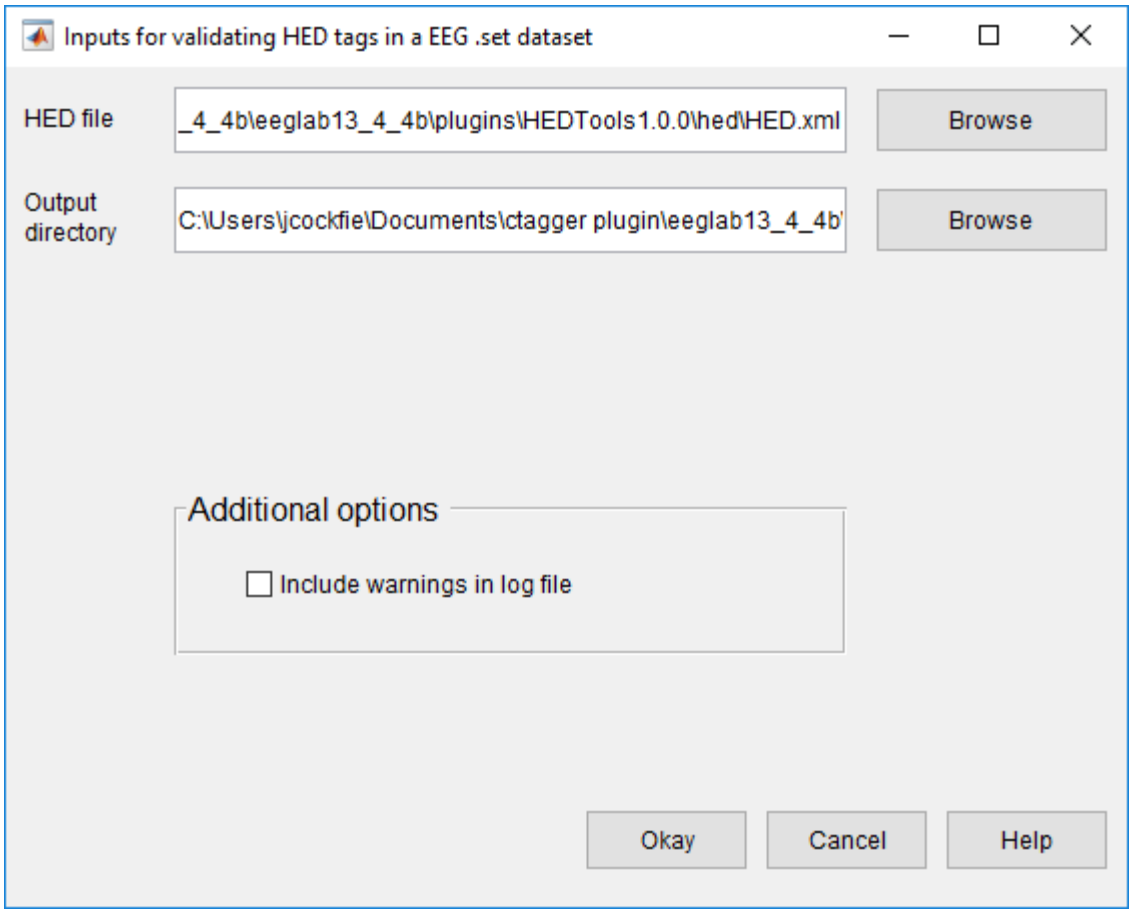

Figure 6. pop_validateeeg menu.

In Figure 6 above, the top section allows you to browse for a HED version 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 3.1: Validate a dataset with additional arguments using a menu.

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

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.

When working exclusively from the command-line you want to use the *validateeeg* function. The following example illustrates the simplest use of *validateeeg* for validating a dataset.

Example 3.2: Validate a dataset and write the output to the workspace and a log file under the current directory.

```
issues = validateeeg(eeg);
```

The *validateeeg* function can also be executed with additional options which are specified in the table 3 below.

Example 3.3: Validate a dataset and only write the output to the workspace.

```
issues = validateeeg(eeg, 'writeOutput', false);
```

Table 4. A summary of arguments for 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 name or the path of the HED XML file containing all of the tags.
`'outDir'`	Name-Value	The directory where the validation output will be written to if the 'writeOutput' argument is true. There will be three separate files generated, one containing the validation errors, one containing the validation warnings, and one containing the extension allowed validation warnings. The default directory will be the directory that contains the tab-delimited text file.
`'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

The *validatedir* function and its supporting functions (*validatedir_input* and *pop_validatedir*) allow you to validate the tags in an entire directory from a script with no user intervention, from the command line with a menu, or from EEGLAB. When run from EEGLAB, you can validate a directory through the EEGLAB *Validate files* submenu under the *File* menu without reading any datasets into EEGLAB. *HEDTools* only considers *.set* datasets. Please convert them to *.set* format if they are not.

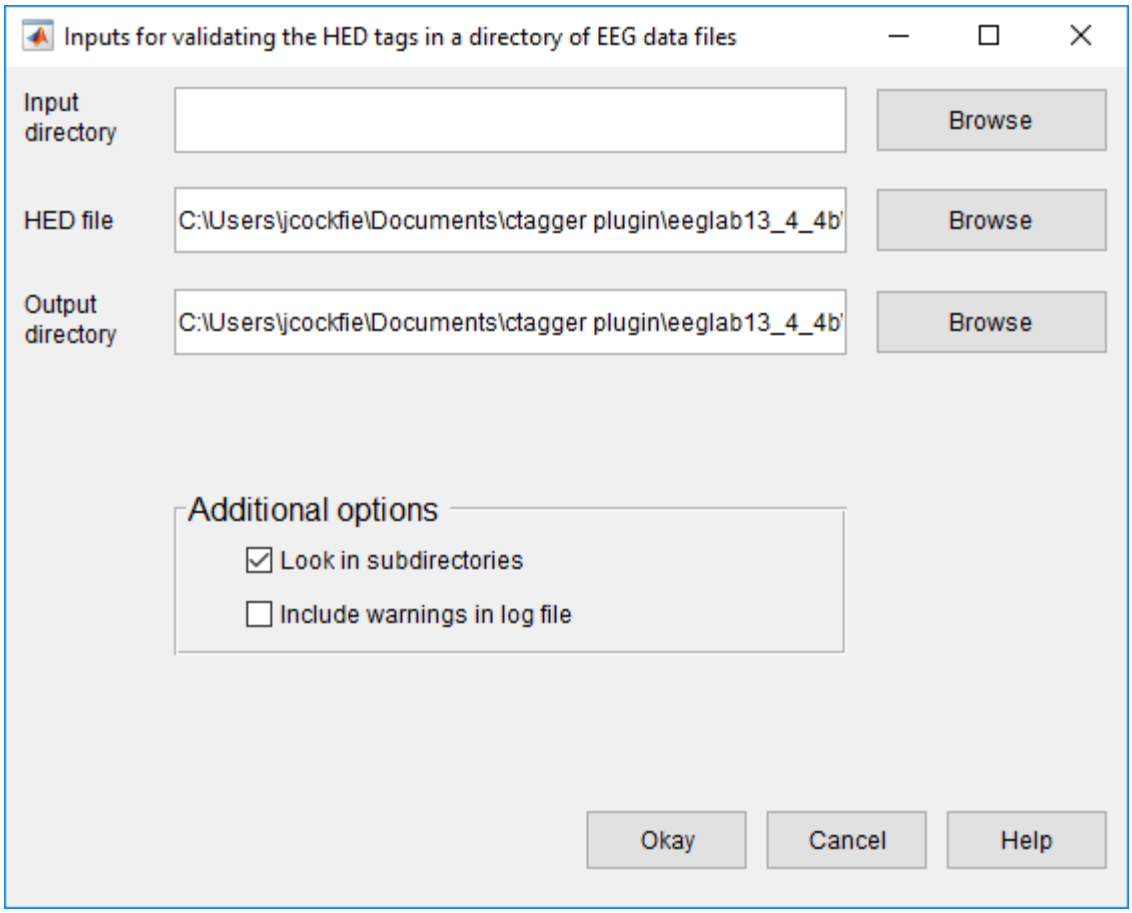

Figure 7. pop_validatedir menu.

In Figure 7 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 3.4: Validate a directory of datasets using a menu.

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

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.

When working exclusively from the command-line you want to use the *validatedir* function. The following example illustrates the simplest use of *validatedir* for validating a dataset.

Example 3.5: Validate a directory of datasets and write the output to the current directory.

```
fPaths = validatedir(inDir);
```

The *validateeeg* function can also be executed with additional options which are specified in the table 3 below.

Example 3.6: Validate a directory of datasets and include warnings in the log files.

```
fPaths = validatedir(inDir, 'generateWarnings', false);
```

Table 5. A summary of arguments for 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

The *validatestudy* function and its supporting functions (*validatestudy_input* and *pop_validatestudy*) allow you to validate the tags in a study from a script with no user intervention, from the command line with a menu, or from EEGLAB. When run from EEGLAB, you can validate a study through the EEGLAB *Validate files* submenu under the *File* menu without reading any datasets into EEGLAB. The study file needs to have a *.study* extension.

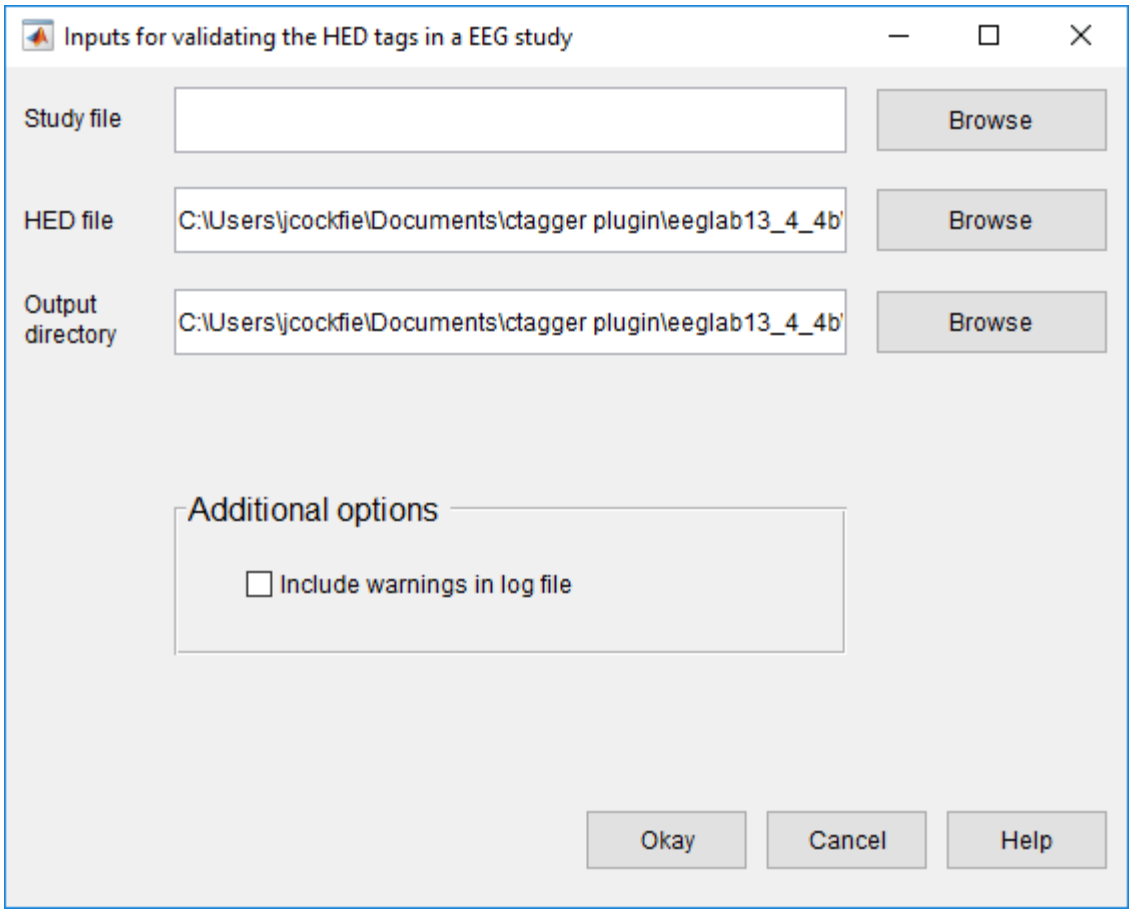

Figure 8. pop_validatestudy menu.

In Figure 8 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 3.7: 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.

When working exclusively from the command-line you want to use the *validatestudy* function. The following example illustrates the simplest use of *validatestudy* for validating a study.

Example 3.8: Validate a study and write the output to the current directory.

```
fPaths = validatestudy(studyFile);
```

The *validatestudy* function can also be executed with additional options which are specified in the table 3 below.

Example 3.9: Validate a study and include warnings in the log files written to the current directory.

```
fPaths = validatestudy(studyFile, 'errorLogOnly', false);
```

Table 6. A summary of arguments for 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_hedepoch* shown in 9 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.

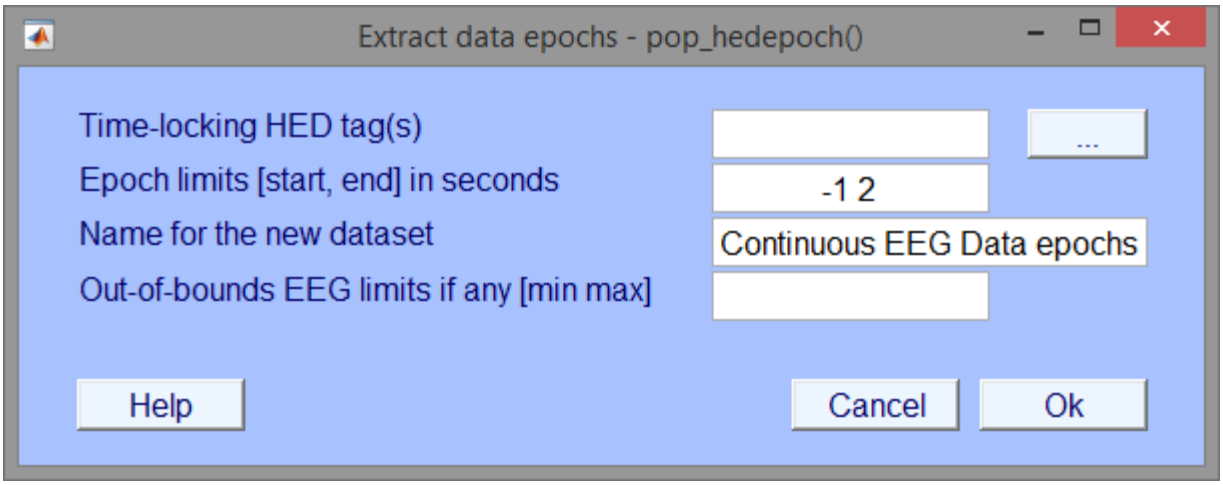

Figure 9. Input settings menu for hedepoch.

The *pop_hedepoch* 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 10.

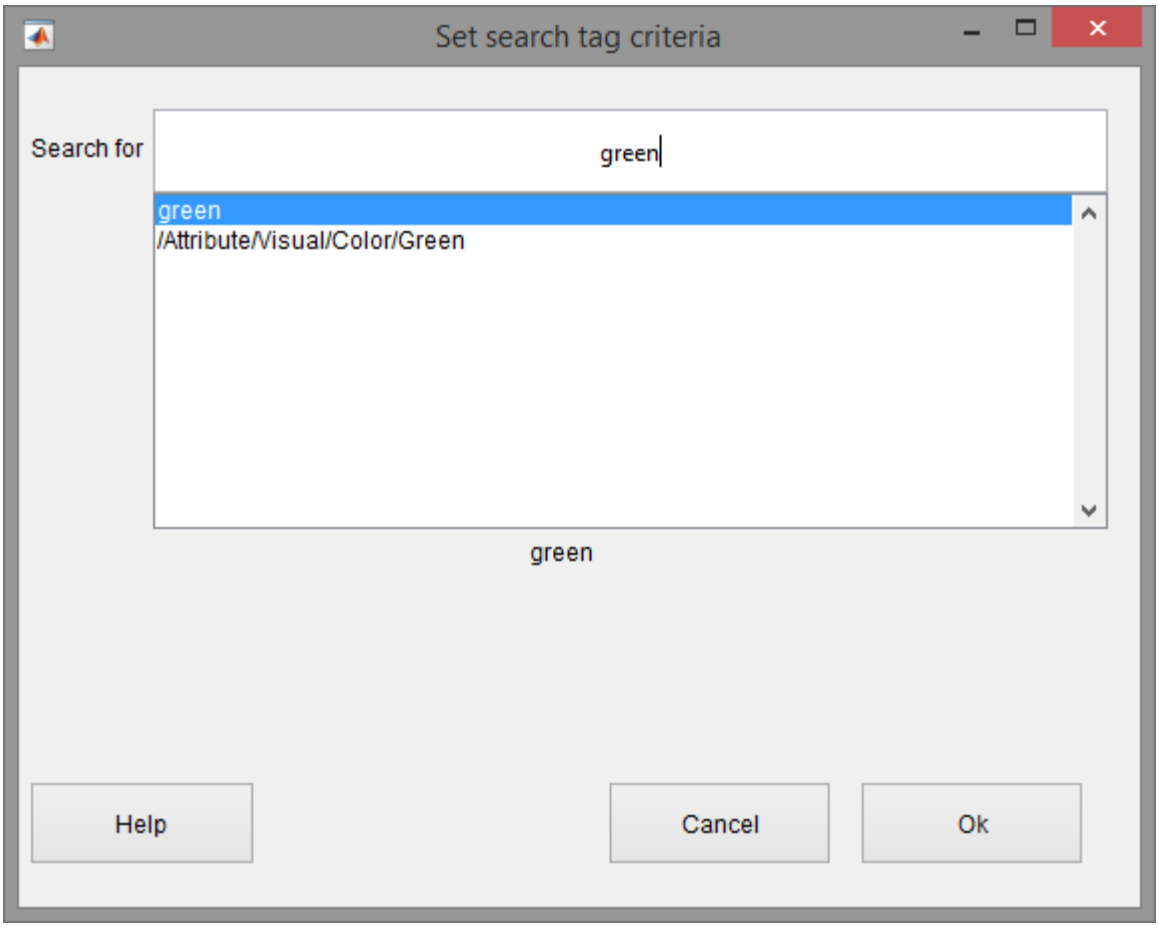

Figure 10. hedepoch search bar.

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]" searchers for events that have a green square or a green circle.

When you type something in the search bar it displays a list 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.

The following example illustrates the simplest use of *hedepoch* (called by *pop_hedepoch*) for extracting data epochs based on HED tags.

Example 4.1: Extract data epochs from events with a green square.

```
EEG = hedepoch(EEG, 'attribute/visual/color/green, item/2d
shape/rectangle/square')
```

Example 4.2: Extract data epochs based on HED tags with menu in Figure 9.

```
[EEG, com] = pop_hedepoch(EEG)
```

MATLAB Syntax

```
EEG = hedepoch(EEG, tags)
EEG = hedepoch(EEG, tags, varargin)
```

Table 7. A summary of arguments for hedepoch.

Name	Type	Description
`EEG`	Required	Input dataset. Data may already be epoched; in this case, extract (shorter) subepochs time locked to epoch events. The dataset is assumed to be tagged and has a .usertags field in the .event structure.
`tags`	Required	A search string consisting of tags to extract data epochs. The 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 will change the order in which the search statements are evaluated. For example, "/attribute/visual/color/green AND [/item/2d shape/rectangle/square OR /item/2d shape/ellipse/circle]".
`'newname'`	Name-Value	New dataset name. The default is "[old_dataset] epochs".
`'timelim'`	Name-Value	Epoch latency limits [start end] in seconds relative to the time-locking event. The default is [-1 2].
`'valuelim'`	Name-Value	[min max] data limits. If one positive value is given, the opposite value is used for lower bound. For example, use [-50 50] to remove artifactual epoch. The default is [-Inf Inf].
`'verbose'`	Name-Value	['on'|'off']. The default is 'on'.

5. Data Formats

Community tagging is structured and hence requires two items: a tag hierarchy and a map of tags to field or group values. The tag hierarchy is in XML format and *HEDTools* provides a schema for validation. The map of tags to field values is in one of four possible formats: a MATLAB structure array, a JSON string, a tab-delimited spreadsheet, or a *tagMap* object. Most of the *HEDTools* functions use the *tagMap* object representation, which provides methods to convert to and from the other representations.

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 XML representation of the tagging menu displayed in Figure. 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 *CTAGGER* functions take an optional *PreservePrefix* argument, which is *false* by default. If you set this argument to *true*, *CTAGGER* 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, *HEDTools* 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 Representing tag maps as a JSON string

JSON (JavaScript Object Notation) is a compact, self-annotating data format that allows objects to be marshaled as strings for passing across network connections. Each JSON library converts a JSON string into a different native format. JSON is light-weight representation used for passing tag information between MATLAB and Java. The jsonlab MATLAB library [2] can translate between JSON strings and MATLAB structures. Our JSON representation of field maps and tag maps is the jsonlab translation of the structures described in the previous section.

The tags associated with each event are represented by a 2D array of strings. An array with a single tag is considered an event-level tag, while an array with multiple tags defines a tag group. In Example 5.7, the second event listed has two tag groups, and the rest are event-level tags. When using this JSON format to represent tagged events, the tagging GUI expects the corresponding tag hierarchy in XML as described above.

Example 5.7: JSON representation of events with tags and tag groups:

```
[
    {
        "code" : "1111",
        "tags" : [
        ["Event/Label/some event 2"],
        ["Event/Description/some event 2's description"],
        ["Event/Long name/Vehicle | Perturbation | Left | Offset"]
        ]
    },
    {
        "code" : "1123",
        "tags" : [
            ["Event/Label/some event 1"],
            ["Event/Long name/Environmental | MissionBoundary |
DataCollection | Onset"],
            ["Event/Description/some event 1's description"],
            ["Sensory presentation/Taste"],
            [
                "Item/Object/Person/Pedestrian",
                "Item/Object/Person/Mother-child",
                "Item/Object/Food"
            ],
            ["Item/3D shape/Sphere"],
            [
                "Item/Object/Animal",
                "Item/Object/Building"
            ]
        ]
    }
]
```

We use two jsonlab functions: *loadjson* and *savejson*. The *loadjson* function converts the JSON string to the MATLAB structure described in example 5.4. The *savejson* function converts the MATLAB structure to JSON.

```
    jStruct = savejson('', jString);
```

```
jString = loadjson(jStruct);
```

5.5 Representing tag maps as tab-delimited text

In a typical experiment, researchers often organize their events in a spreadsheet and an XML representation is not convenient or understandable. *HEDTools* supports loading of events and their annotations in a tab-delimited text format into *a tag map* shown in example 5.8 for this purpose. Each row or line of the file represents an event, with the event code or event latency in particular columns and tags optionally in other columns.

Within the tag columns, tags appear as text strings separated by commas. Define a tag group by placing parentheses around a comma-separated list of tags. Example 5.8 shows an example with three events (with lines wrapped to fit). The event codes are 1111, 1121, and 1112, respectively. Event code 1111 indicates the start of a perturbation of a car to the left in a driving experiment.

Example 5.8: Tab-delimited text representation of events

```
Event      HED Tags
1111        Event/Category/ExperimentalStimulus,
Event/Label/LeftPerturbOnset, (Item/Object/Vehicle/Car,
Attribute/Object control/Perturb, Attribute/Direction/Left)
1121        Event/Category/ExperimentalStimulus,
Event/Label/RightPerturbOnset, (Item/Object/Vehicle/Car,
Attribute/Object control/Perturb, Attribute/Direction/Right)
1112        Event/Category/ExperimentalStimulus,
Event/Label/LeftPerturbOffset, (Item/Object/Vehicle/Car,
Attribute/Object control/Perturb, Attribute/Direction/Left)
```

As mentioned above tab-delimited text format can be converted to a tag map. To do this you must call the *tagtsv* function.

```
tsvTagMap = tagtsv(filename, fieldname, eventColumn, tagColumns)
```

The *filename* is the name (including the path) of the tab-delimited file containing the events and their tags. The *fieldname* argument is the unique field that is associated with the event column values. For Example 6.8 the *fieldname* would be *'type'* which describes the different kinds of events in the file. The *eventColumn* and *tagColumns* are the column(s) that contain the events and tag respectively. The *eventColumn* is a single integer value while *tagColumns* can be a single integer value or a vector of integer values. When no tags are present in the file, pass in an empty vector ([]) for the *tagColumns* argument. The output argument is *tsvTagMap* which is a tag map encapsulating all of the events.

5.6 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 5.9: 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 5.10: 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 of a *stimpos* value *1* are consolidated in *EEG.event(1).usertags* to allow data-mining. *HEDTools* 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.

5.7 The *fieldMap* object

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

Example 5.11: 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(varargin)
```

Table 8. 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 9. 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 10. 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).

5.8 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 5.12: Representation of *fMap.map(1)* of Example 5.10 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(varargin)
```

Table 11. A summary of arguments for tagMap constructor.

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

Table 12. 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 13. 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.

5.9 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 5.13: 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 14. A summary of arguments for tagList constructor.

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

Table 15. 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 16. 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. Saving tags in the dataset (the `writetags` function)

All of the higher-level functions call the *writetags* function to write the tag information to the dataset. *HEDTools* writes the tags in two different ways: as a summary field map in the *.etc.tags* subfield of the data and as individual event information. In the latter situation, it's assumed that the events to be tagged are in stored in the *.event* structure array and it writes a consolidated list of tags based on the actual values of different fields for the i^{th} event to the *.event(i).usertags* subfield.

Example 6.1: Write the tags encapsulated by the *fieldMap* object *fMap* into the data structure `x`.

```
x = writetags(x, fMap);
```

The *writetags* writes both the summary and individual event information, overwriting existing tagging information. If `x` doesn't have an *.event* structure, no individual event information is written. The *fMap* object can come from anywhere. Thus, you can have multiple tagging schemes and merge them before writing, or use one at a time. An advantage of keeping the mappings as summaries, separate from the events is that you can edit your tags and rewrite for different uses.

MATLAB Syntax

```
eData = writetags(eData, fMap)
eData = writetags(eData, fMap, varargin)
```

Table 17. A summary of arguments the writetags function.

Name	Type	Description
`eData`	Required	A dataset structure that tag information is to be written to.
`fMap`	Required	A *fieldMap* object with the tag information.
`'ExcludeFields'`	Name-Value	A cell array of field names in the *.event* and *.urevent* substructures to ignore during rewrite.
`'PreservePrefix'`	Name-Value	If *false* (default), *HEDTools* combines tags of the same event type that share prefixes and only retains the most specific (e.g., /a/b/c and /a/b become just /a/b/c). If *true*, *HEDTools* retains all unique tags.

7. Running the regression tests and examples

CTAGGER uses the XUNIT unit-testing framework for its regression tests located in the *tests* directory. For tests that require user input, the instructions appear in caps in the command window. The regression tests use external data not located in the HEDTools repository. Download the test data archive (*HEDToolsTestArchive.zip*) and unzip it. You will also need to edit the *tests/setup_tests.m* file and adjust the *values.testroot* to contain the path of your unzipped archive. *CTAGGER* comes with examples contained in the *tagging_example.m* script. The examples use external data not located in the HEDTools repository. Download the example data archive (*HEDToolsExampleArchive.zip*) and unzip it. You will also need to edit the *exampleDir* that points to the example data archive.

8. Status and availability

The base *HEDTools* is currently available and undergoing user testing. Also, the community tagging database is currently being developed and tested, but will not be available for this release.

9. 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.

10. 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].