CTAGGER: Community Tagging Tools User Guide

By

Thomas Rognon, Rebecca Strautman, Lauren Jett, Jeremy Cockfield and Kay Robbins

Department of Computer Science

University of Texas at San Antonio

June 2013

Updated: July 14, 2013, October 2, 2013, October 12, 2013

Table of Contents

1. Getting Started with *CTAGGER* Tools

1.1 Overview

Community tagging, also known folksonomy or collaborative tagging refers to systems in which groups of users assign tags to data elements in order to cooperatively annotate and categorize content. The tags facilitate searching, data mining, and data sharing. Community tagging has many applications, particularly in large-scale annotation and metadata generation for large scientific databases.

CTAGGER is a MATLAB/Java Toolbox that facilities user tagging. The toolbox has functions that allow users to create tag maps to associate event-types with tags, enabling large-scale analysis of multi-modal event-rich data collections. The underlying infrastructure of *CTAGGER* is written in Java, but accessed through MATLAB functions. These installation instructions assume that you will not be working with the Java source, but are using the JAR files that come with the *CTAGGER* distribution. The Java source, in the form of an Eclipse project, is available separately.

1.2 Requirements

The *CTAGGER* tools have been tested with MATLAB 2012a and PostgreSQL 9.2 on systems running either 64-bit Windows 7 or Ubuntu 12.04LTS. No other toolboxes are required. If your datasets are EEGLAB `.set` files, then you will also need version EEGLAB 10 or later in order to read the datasets for tagging. **If you are not running your own database of community tags, you will not need to install PostgreSQL.**

1.3 Installation

CTAGGER can be run 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 and you aren't maintaining your own database, you can simply unzip the *CTAGGER* toolbox anywhere you choose. Execute the `setup.m` script to set the paths each time you enter MATLAB. Alternatively, you can add the code contained in `setup.m` to your `startup.m` script. If you are not using EEGLAB, you can comment out the last section of the `setup.m` script.

If you want to access a database of community tags and you have the credential information (server location, database name, user name, and password), you can use the `dbcreds.m` function to create a property file containing your credentials. This property file is needed if you want to access the database.

1.3.2 Running with data file types that include .set files

If you want to tag data stored in EEGLAB `.set` files as well as MATLAB `.mat` files, then EEGLAB should be in your path. Follow the directions above, but instead of commenting out the last section of the `setup.m` script, you should either unzip the *CTAGGER* toolbox in the `plugins` directory of EEGLAB or modify the `EEGLAB_PATH` variable in the `setup.m` script to point to the location of EEGLAB for your installation. As before, execute the `setup.m` script to set the paths each time you enter MATLAB or add the `setup.m` code to your `startup.m` script.

1.3.3 Running as a plugin to EEGLAB

CTAGGER has three options for tagging from the EEGLAB menus: tagging the current EEG (from the *Edit* menu), tagging the current STUDY (from the *Study* menu), or tagging a directory (from the *File* menu). To *CTAGGER* use as a plug-in, unzip the *CTAGGER* toolbox so that the `ctagger` directory is directly under the EEGLAB `plugins` directory. When you bring up EEGLAB, the paths will automatically be set up.

1.4 Running with a community tagging database

In order for *CTAGGER* to truly behave as a community tagging system, it must be run with a back-end database that stores a common tag hierarchy with usage counts for the individual tags. The purpose of the database is to benefit from additions to the hierarchy by other users and to encourage use of common tags to facilitate data mining across data collections. If you are not running your own community tagging database, you do not need to install and maintain your own PostgreSQL database. To use a database to load and store the tag information, you will need to provide the following information in standard property file format:

Table 1.1: Database credentials in property file.

Parameter	Description
`hostname`	A string giving the URL or IP address of the computer running the database server. If you are running a database on your local machine, use `'localhost'`.
`port`	An integer giving the database server port number. The default for PostgreSQL is `'5432'`.
`dbname`	The name of the database you are using.
`username`	The user name of the database account you are using.
`password`	The password of the database account you are using.

You can run the `dbcreds.m` script to create and save your credential file. The top-level tagging functions (such as `tagdir`) take an optional `'DbCreds'` argument. If you provide a valid credentials file using this parameter in your function call, the call will transparently load the current tag hierarchy (with the counts) from the database and store the results in the database when you submit.

If you are running your own community database backend, you will need to install a PostgreSQL server and create a *CTAGGER* database. **Most users will not want to do this individually.** The benefit of community tagging comes from collaborating with a group of users on a common tagging hierarchy. Once you have installed PostgreSQL, simply run the `setupdb.m` script after you have run the `setup.m` script.

Install PostgreSQL by downloading and executing the appropriate installer for your platform (e.g., Windows 64 bit, Linux, etc.). You can follow the default installation with the following exceptions:

a) The administrator account for the server is *postgres*. You will be asked for a password for the database administrator. The examples in this guide use the password *admin*, but you should choose (and remember) a more secure password. Make note of the PostgreSQL installation and data directories, as well as the port number for the server.

b) **Do not press the Finish button, until you have unchecked the option to install items using the stackbuilder**. Some items in the stack builder cause the database to perform very slowly.

c) The pgAdmin3 GUI for administering the PostgreSQL server comes with the distribution in the `bin` subdirectory of your PostgreSQL installation. This tool is a very handy companion for monitoring the databases that you create. You probably should create a shortcut on your desktop for this tool so that you can monitor the various databases and database servers that you are working with.

2. Tagging a single EEGLAB EEG structure

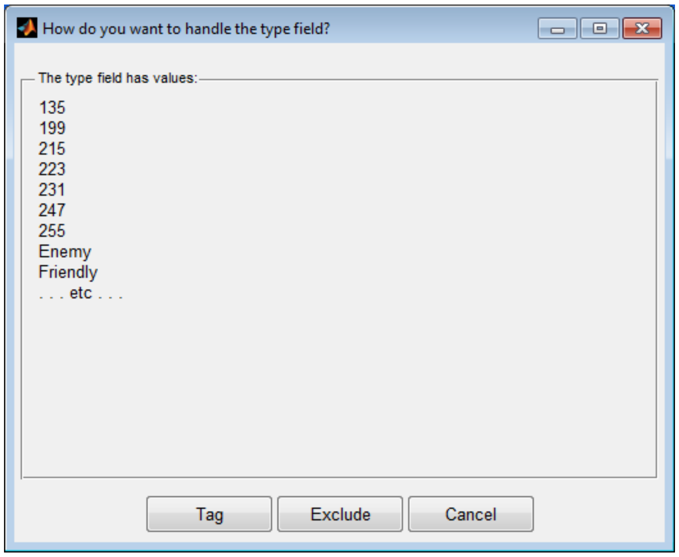
The *CTAGGER* `tageeg` function and its supporting functions (`tageeg_input` and `pop_tageeg`) allow you to tag a single EEGLAB `EEG` structure either from a script with no user intervention, from the command line with GUI input, or from EEGLAB. When run from EEGLAB, you can tag the current `EEG` structure through the EEGLAB *Edit* menu. The `EEG` input data only needs to be a structure and does not have to conform to EEGLAB requirements.

The following example illustrates the simplest use of `tageeg` for interactive tagging of an EEG structure. The input `EEG` structure can be any structure, but field and tag information is only extracted from the `.etc.tags`, the `.event`, and the `.urevent` fields. If the data does not have these fields, the corresponding extraction is skipped.

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

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

The `tageeg` function extracts any tag information already stored in the `EEG` structure and uses this information to populate a `fieldMap` object. You can then select the fields to tag by displaying a series of dialogs. Each sample dialog allows you to decide whether a particular field (group name) should be tagged or excluded. The dialog displays up to 10 of the unique values for each field or group and asks whether this field should be tagged or excluded. The example dialog in Fig. 2.1 shows the `.type` field for a sample dataset. If you press the *Tag* button, the `.type` field group will be included in the tag map. If you press the *Exclude* button, the `.type` field will not be included.


How do you want to handle the type field?

The type field has values:

135
199
215
223
231
247
255
Enemy
Friendly
. . . etc . . .

Figure 2.1: Example of the selection dialog for the `.type` field or group.

Once you have picked the fields to be tagged, the `tageeg` function displays the tagging GUI for each selected field as shown in Fig. 2.2 for the `.type` field of Fig. 2.1. After tagging is complete, `tageeg` writes the tag information to the `.etc.tags` and `.event.usertags` fields of the return `EEG` structure. The `fMap` return argument is a `fieldMap` object that contains the tag map information created during this call.

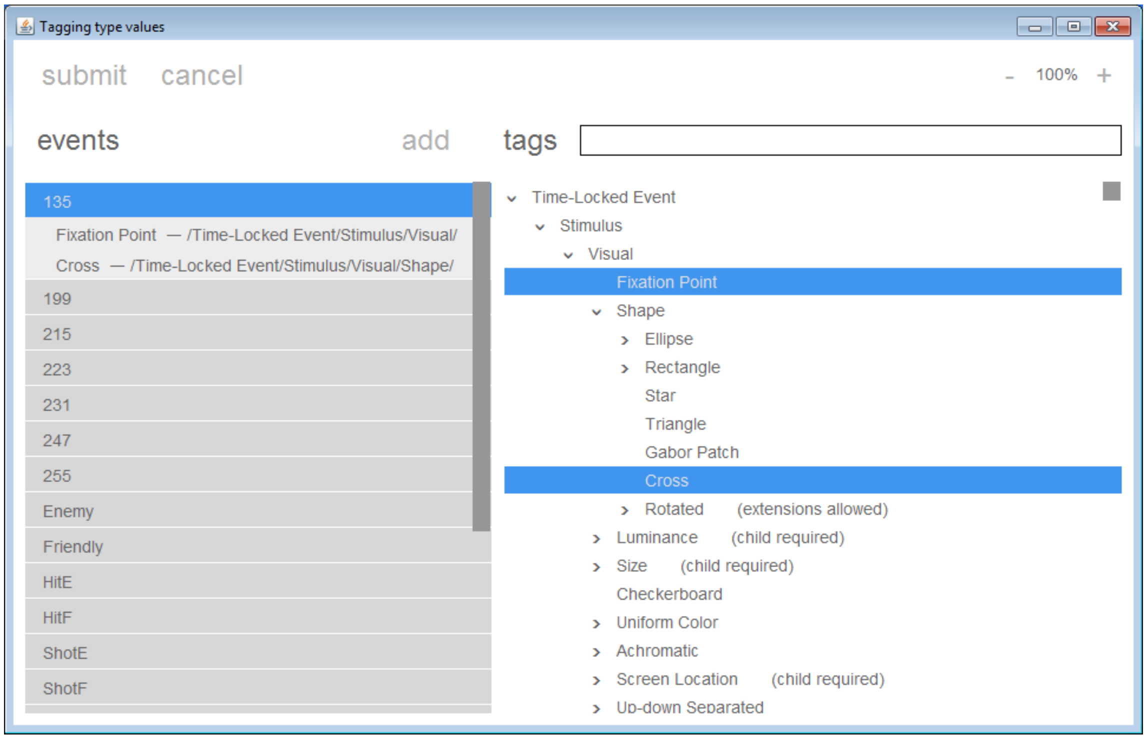


Figure 2.2: Example of the tagging GUI for the `.type` field or group.

In Fig. 2.2, only the type value 135 has been tagged so far. The GUI allows the users to associate tags with each event (group or field) value. The key idea is that rather than having to choose tags at random, you are presented with a menu of potential tags organized in a hierarchical format from general to more specific. You can annotate your event types by choosing tags from this hierarchical menu. You can also modify the hierarchy by adding tags in the hierarchy. Depending on how the GUI is called, you may also delete or edit existing nodes.

To tag an event on the left, simply click on it to highlight. Tags already selected for that event will be highlighted. Click on tags on the right to select or deselect. You can add events using the add button on the left. You can edit any item by right clicking to bring up a context menu. You can also search for tags. Clicking on a search result causes it to be added or removed from the currently selected event. You can move the tag view to that search tag by right clicking to bring up a menu.

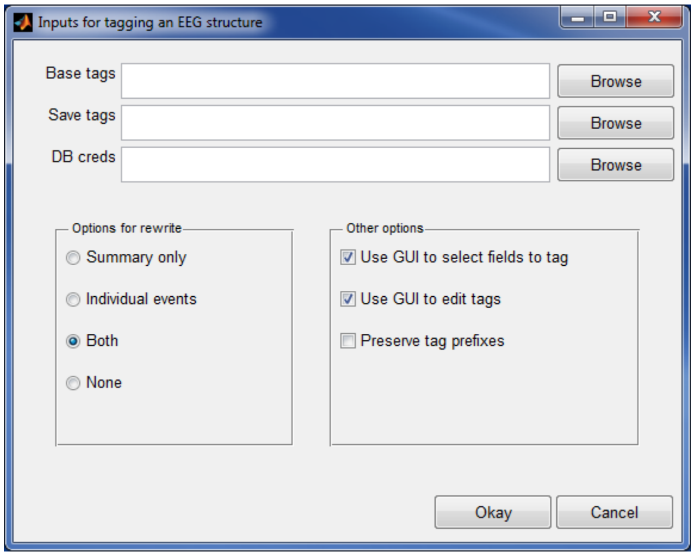
The *CTAGGER* tools also provide a GUI that allows you to set the input parameters using the `pop_tageeg` function.

Example 2.2: Tag an `EEG` structure using a GUI for setting the input arguments of `tageeg`.

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

The `EEG` return value contains the input `EEG` structure, modified by tagging actions. The `com` return parameter contains the command string that you would have typed to execute `tageeg` without the input GUI.

Fig. 2.3 shows the input GUI that allows you to set the arguments as part of the analysis of Example 2.2. The top section of the GUI allows you to browse and select the location of a previously saved `fieldMap` (*Base tags*), a location to save the output `fieldMap` (*Save tags*), and the location of the database credential file if you are using community tagging (*DB creds*). The left box below allows you to specify how to write the tagging results back to your data. The right box below allows you to select other options, such as whether you want to use the selection and tagging GUIs.



Base tags
Save tags
DB creds

Options for rewrite
Summary only
Individual events
● Both
None

Other options
✓ Use GUI to select fields to tag
✓ Use GUI to edit tags
Preserve tag prefixes

Figure 2.3: The input argument GUI (`tageeg_input`) for the `tageeg`.

Example 2.3: Tag another dataset without user intervention using tag map information of Example 2.1.

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

The `'SelectOption'` controls whether or not you are presented with dialogs to select which fields to tag. The `'UseGui'` controls whether or not you see a tagging GUI for each selected field. In Example 2.3 all user intervention is turned off. The idea here is that you tag one dataset and then call `tageeg` without user intervention to tag related datasets with no additional work. Similarly, the `tagdir` function consolidates the tag information of all of the datasets in a directory, allows the user to select fields and tag the consolidated information, and then writes the information back to the datasets.

MATLAB Syntax

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

Table 2.1: 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.
`'DbCreds'`	Name-Value	Name of a property file containing the database credentials. If this argument is not provided, a database is not used.
`'ExcludeFields'`	Name-Value	A cell array of field names in the `.etc.tags`, `.event,` and `.urevent` substructures 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 exclusions using this name-value parameter.
`'Fields'`	Name-Value	A cell array of field names of the fields to include in the tagging. If this parameter is non-empty, only these fields are tagged, provided they have not been excluded.
`'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.
`'RewriteOption'`	Name-Value	A string indicating how tag information should be written to the datasets. The options are `'Both'`, `'Individual'`, `'None'`, and `'Summary'`. (See the section on rewriting tags to datasets for additional explanation.)
`'SaveMapFile'`	Name-Value	The full path name of the file for saving the final, consolidated `fieldMap` object that results from the tagging process.
`'SelectOption'`	Name-Value	If true (default), the user is presented with dialog GUIs that allow them to select which fields to tag.
`'Synchronize'`	Name-Value	If false (default), the *CTAGGER* GUI is run with synchronization done using the MATLAB `pause`. If true, synchronization is done within Java. This latter option is usually reserved when not calling the GUI from MATLAB.
`'UseGui'`	Name-Value	If true (default), the *CTAGGER* GUI is displayed after initialization.

3. Tagging a directory of datasets

The *CTAGGER* `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 GUI input, or from EEGLAB. When run from EEGLAB, you can tag a directory through the EEGLAB *File* menu without reading any datasets into EEGLAB.

The following example illustrates the simplest use of `tagdir` for interactive tagging of a directory.

Example 3.1: Tag the data in the `inDir` directory.

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

The `tagdir` function extracts tag information from all of the datasets stored in the `inDir` directory tree and uses this information to populate a `fieldMap` object. By default, only `.set` datasets are considered. However, this is configurable. You will then be presented with dialogs, similar to the one of Fig. 2.1, allowing you to decide whether particular fields (group names) should be tagged or excluded.

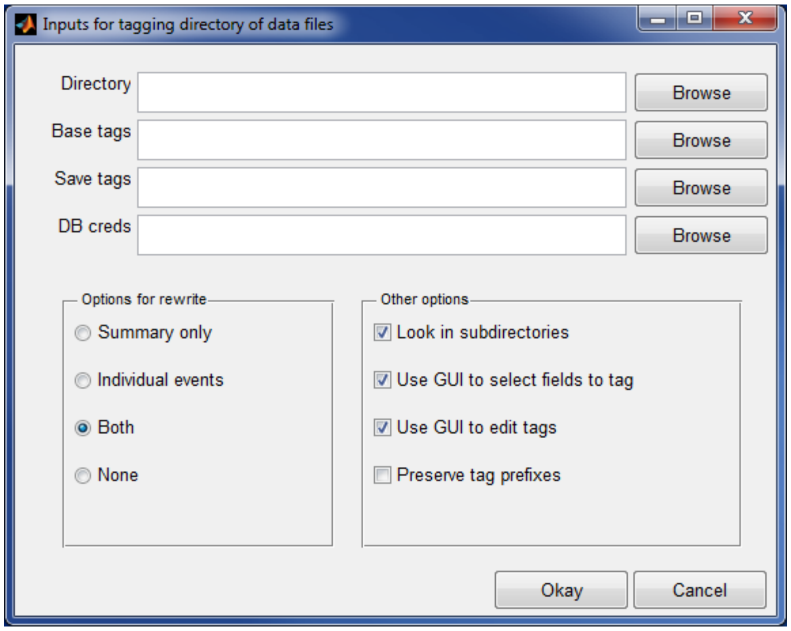
Once you have picked the fields to be tagged, the `tagdir` function displays the tagging GUI (see Fig. 2.2) for each selected field. After you have completed tagging, `tagdir` writes the tag information back to the datasets. The `fMap` return argument is a `fieldMap` object that contains the tag map information created during this call. The `fPaths` return argument is a cell array containing the full path names of the datasets tagged during this call. The `excluded` argument is a cell array of field names of excluded fields.

The *CTAGGER* tools also provide a GUI that allows you to set the input parameters using the `pop_tagdir` function.

Example 3.2: Tag a directory using a GUI to enter the input of arguments for `tagdir`.

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

Fig. 3.1 shows the input GUI that allows you to set the arguments as part of the analysis for Example 3.2. The top section of the GUI allows you to browse and select the location of the directory tree for the datasets to be tagged (*Directory*), a previously saved `fieldMap` containing initial tags (*Base tags*), a location to save the output `fieldMap` (*Save tags*), and a file name for database credentials if you are using community tagging (*DB creds*). The left box below allows you to specify how to write the tagging results back to your data. The right box below allows you to select other options, such as whether you want to use the selection and tagging GUIs.


Inputs for tagging directory of data files

Directory	
Base tags	
Save tags	
DB creds	

Options for rewrite
- Summary only
- Individual events
- [✓ Both]
- None

Other options
- [✓ Look in subdirectories]
- [✓ Use GUI to select fields to tag]
- [✓ Use GUI to edit tags]
- Preserve tag prefixes

Figure 3.1: The input argument GUI (`tagdir_input`) for the `tagdir`.

Example 3.3: Tag a directory without user intervention using the tag map information of Example 3.2.

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

The `'SelectOption'` controls whether you are presented with dialogs to select which fields to tag. The `'UseGui'` controls whether or not you see a tagging GUI for each selected field. In Example 3.3 all user intervention is off. The idea here is that you tag an entire directory without user intervention once you have created your tag mapping.

MATLAB Syntax

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

Table 3.1: A summary the of arguments for `tagdir`.

Name	**Type**	**Description**
`inDir`	Required	The full path name of the directory to be tagged.
`'BaseMap'`	Name-Value	A `fieldMap` object or the name of a file that contains a `fieldMap` object to be used for initial tag information.
`'DbCreds'`	Name-Value	The full path name of a property file containing the database credentials. If this argument is not provided, a database is not used.
`'DoSubDirs'`	Name-Value	If true (default), the entire `inDir` directory tree is searched. If false, only the `inDir` directory itself is searched.
`'ExcludeFields'`	Name-Value	A cell array of field names in the `.etc.tags`, `.event,` and `.urevent` substructures 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 exclusions using this name-value parameter.
`'Fields'`	Name-Value	A cell array of field names of the fields to include in the tagging. If this parameter is non-empty, only these fields are tagged provided they have not been excluded.
`'PreservePrefix'`	Name-Value	If false (default), tags of 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.
`'RewriteOption'`	Name-Value	A string indicating how tag information should be written to the datasets. The options are `'Both'`, `'Individual'`, `'None'`, and `'Summary'`. See the section on rewriting tags to datasets for additional explanation.
`'SaveMapFile'`	Name-Value	The full path name of the file for saving the final, consolidated `fieldMap` object that results from the tagging process.
`'SelectOption'`	Name-Value	If true (default), the user is presented with dialog GUIs that allow them to select which fields to tag.
`'Synchronize'`	Name-Value	If false (default), the *CTAGGER* GUI is run with synchronization done using the MATLAB `pause`. If true, synchronization is done within Java. This latter option is usually reserved when not calling the GUI from MATLAB.
`'UseGui'`	Name-Value	If true (default), the *CTAGGER* GUI is displayed after initialization.

4. Tagging an EEGLAB study

The *CTAGGER* `tagstudy` function and its supporting functions (`tagstudy_input` and `pop_tagstudy`) allow you to tag an EEGLAB study from a script with no user intervention, from the command line with GUI input, 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 `tagstudy` for interactive tagging of an EEGLAB study.

Example 4.1: Tag the data represented by the EEGLAB study specified by the `studyFile` file.

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

The `tagstudy` function extracts any tag information from the study file and associated datasets, using this information to populate a `fieldMap` object. You will then be presented with a series of dialogs, similar to the one of Fig. 2.1. These dialogs allow you to decide whether a particular field (group name) should be tagged or excluded.

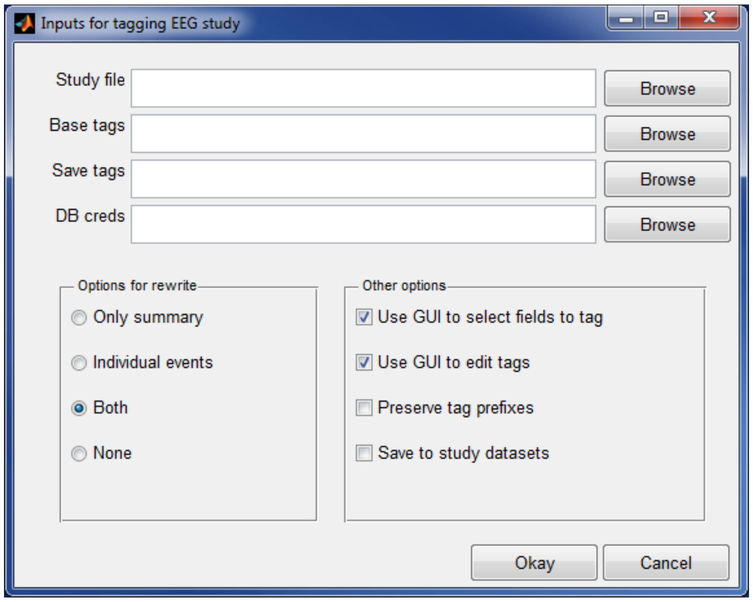
Once you have picked the fields to be tagged, the `tagstudy` function displays the tagging GUI (see Fig. 2.2) for each selected field. After you have completed your tagging, `tagstudy` writes the tag information back to the datasets and the study file, depending on the rewrite options that you have selected. The `fMap` return argument is a `fieldMap` object that contains the tag map information created during this call. The `fPaths` return argument is a cell array containing the full path names of the datasets associated with the study. The `excluded` argument is a cell array of field names of excluded fields.

The *CTAGGER* tools also provide a GUI that allows you to set the input parameters using the `pop_tagstudy` function.

Example 4.2: Tag a study using a GUI for input.

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

Fig. 4.1 shows the input GUI that allows you to set the arguments as part of the analysis for Example 4.2. The top section of the GUI allows you to browse and select the location of the study file for the datasets to be tagged (*Study file*), a previously saved `fieldMap` containing initial tags (*Base tags*), a location to save the output `fieldMap` (*Save tags*), and a file name for database credentials if you are using community tagging (*DB creds*). The left box below allows you to specify how to write the tagging results back to your data. The right box below allows you to select other options, such as whether you want to use the selection and tagging GUIs.


Inputs for tagging EEG study

Study file [Browse]
Base tags [Browse]
Save tags [Browse]
DB creds [Browse]

Options for rewrite: Only summary, Individual events, [✓ Both], None

Other options: [✓ Use GUI to select fields to tag], [✓ Use GUI to edit tags], Preserve tag prefixes, Save to study datasets

Figure 4.1: The input argument GUI (`tagstudy_input`) for the `tagstudy`.

Example 4.3: Tag a study without user intervention using tag map information of Example 4.2.

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

The `'SelectOption'` controls whether you are presented with dialogs to select which fields to tag. The `'UseGui'` controls whether or not you see a tagging GUI for each selected field. In Example 4.3 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 4.1: A summary of arguments for `tagstudy`.

Name	Type	Description
`studyFile`	Required	The full path name of the study file for study to be tagged.
`'BaseMap'`	Name-Value	A `fieldMap` object or the name of a file that contains a `fieldMap` object to be used for initial tag information.
`'DbCreds'`	Name-Value	Name of a property file containing the database credentials. If this argument is not provided, a database is not used. (See notes.)
`'ExcludeFields'`	Name-Value	A cell array of field names in the `.etc.tags`, `.event,` and `.urevent` substructures 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 exclusions using this name-value parameter.
`'Fields'`	Name-Value	A cell array of field names of the fields to include in the tagging. If this parameter is non-empty, only these fields are tagged provided they have not been excluded.
`'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.
`'RewriteOption'`	Name-Value	A string indicating how tag information should be written to the datasets. The options are `'Both'`, `'Individual'`, `'None'`, `'Summary'`. See the section on rewriting tags to datasets for additional explanation.
`'SaveMapFile'`	Name-Value	The full path name of the file for saving the final, consolidated `fieldMap` object that results from the tagging process.
`'SelectOption'`	Name-Value	If true (default), the user is presented with dialog GUIs that allow users to select which fields to tag.
`'Synchronize'`	Name-Value	If false (default), the *CTAGGER* GUI is run with synchronization done using the MATLAB pause. If true, synchronization is done within Java. This latter option is usually reserved when not calling the GUI from MATLAB.
`'UseGui'`	Name-Value	If true (default), the *CTAGGER* GUI is displayed after initialization.

5. Tagging CSV files

The *CTAGGER* `tagcsv` function and its supporting functions (`tagcsv_input` and `pop_tagcsv`) allow you to tag a comma separated value (csv) file from a script with no user intervention, from the command line with GUI input, or from EEGLAB. When run from EEGLAB, you can tag a csv file through the EEGLAB *File* menu without reading any datasets into EEGLAB.

The following example illustrates the simplest use of `tagcsv` for interactive tagging of a csv file.

Example 5.1: Tag the data in the file specified by filename.

```
fMap = tagcsv(filename);
```

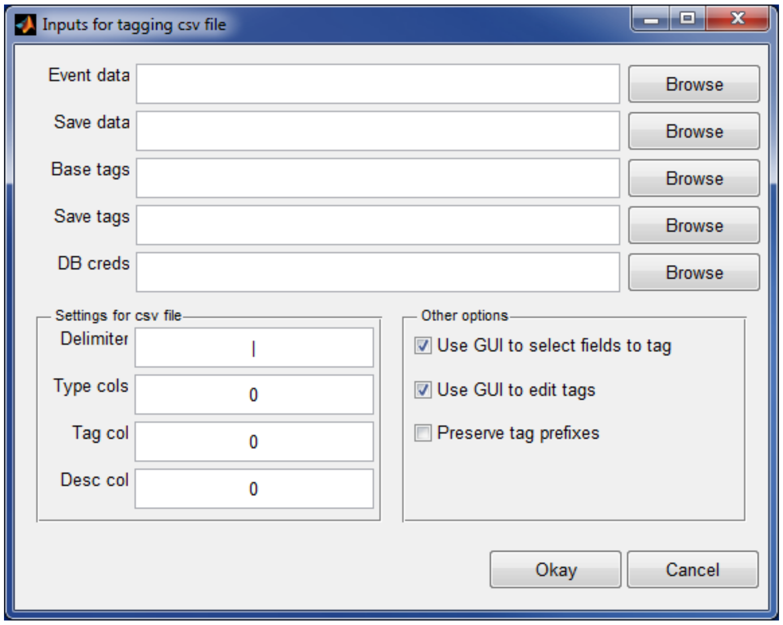
The `tagcsv` function extracts tag information from columns in the input csv file and uses this information to populate a `fieldMap` object. Since no columns or delimiters have been specified, *CTAGGER* assumes that all of the columns specify event types and appends all of the column values in each row using a vertical bar (`'|'`) as a separator to form a string. For example, if the csv file has three columns, *CTAGGER* assumes a row with entries (`'apple', '342.0', '78'`) represents an event type `'apple|342.0|78'`. The event columns cannot contain the delimiter character and should also not contain commas. The `tagcsv` function then displays the tagging GUI (see Fig. 2.2). After you complete the tagging process, `tagcsv` returns a `fieldMap` object `fMap` that contains the tag map information created during this call. Since the command doesn't specify a file to save the results in csv format, `tagcsv` doesn't produce any output files.

The *CTAGGER* tools also provide a GUI that allows you to set the input parameters using the `pop_tagcsv` function.

Example 5.2: Tag a csv file using a GUI for input of arguments for `tagcsv`.

```
[fMap, com]= pop_tagcsv();
```

Fig. 5.1 shows the input GUI that allows you to set the arguments as part of the analysis for Example 5.2. The top section of the GUI allows you to browse and select the location of the file that has the events (*Event data*), a filename to use in rewriting the information in csv (*Save data*), a previously saved `fieldMap` containing initial tags (*Base tags*), a location to save the output `fieldMap` (*Save tags*), and a file name for database credentials if you are using community tagging (*DB creds*). The left box below allows you to set the delimiter and the columns of the csv file used for various purposes. The type column specification can be a list of values. For example, `3, 1, 4` specifies that the third, first, and fourth columns should be concatenated using the delimiter to form the specified event type. If a csv file row contained (`'apple', '342.0', '78', 'a'`), the event type would be `'78|apple|a'`. The right box below allows you to select other options, such as whether you want to use the selection and tagging GUIs.


Inputs for tagging csv file

Event data | Browse

Save data | Browse

Base tags | Browse

Save tags | Browse

DB creds | Browse

Settings for csv file

Delimiter: |

Type cols: 0

Tag col: 0

Desc col: 0

Other options

[✓] Use GUI to select fields to tag

[✓] Use GUI to edit tags

[ ] Preserve tag prefixes

Okay | Cancel

Figure 5.1: The input argument GUI (`tagcsv_input`) for the `tagcsv`.

Example 5.3: Tag a csv file without user intervention using the tag map information of Example 5.2.

```
fMap1 = tagcsv(filename, 'BaseMap', fMap, ...  'UseGui', false);
```

The `'UseGui'` controls whether or not you see a tagging GUI. In Example 5.3 all user intervention is off. The idea here is that you tag a csv file without user intervention once you have created your tag mapping.

MATLAB Syntax

```
fMap = tagcsv(filename)
fMap = tagcsv(filename, varargin)
```

Table 5.1: A summary the of arguments for `tagcsv`.

Name	Type	Description
`filename`	Required	The full path name of the csv file to be tagged.
`'BaseMap'`	Name-Value	A `fieldMap` object or the name of a file that contains a `fieldMap` object to be used for initial tag information.
`'DbCreds'`	Name-Value	The full path name of a property file containing the database credentials. If this argument is not provided, a database is not used.
`'Delimiter'`	Name-Value	A string containing the delimiter separating event code components.
`'DescriptionColumn'`	Name-Value	A non-negative integer specifying the column that corresponds to the event code description. Users should provide detailed documentation of exactly what this code means with respect to the particular experiment. If the value is 0, no description is read or written.
`'EventColumns'`	Name-Value	Either a non-negative integer or a vector of positive integers specifying the column(s) that correspond to event code components. If the value is 0, it is assumed that all columns correspond to event codes.
`'PreservePrefix'`	Name-Value	If false (default), tags of the same field value that share prefixes are combined and only the most specific is retained (e.g., /a/b/c and /a/b just become /a/b/c). If true, then all unique tags are retained.
`'RewriteFile'`	Name-Value	File name of the tagged csv file with the result of tags and description written in the file.
`'SaveMapFile'`	Name-Value	The full path name of the file for saving the final, consolidated `fieldMap` object that results from the tagging process.
`'Synchronize'`	Name-Value	If false (default), the *CTAGGER* GUI is run with synchronization done using the MATLAB `pause`. If true, synchronization is done within Java. This latter option is usually reserved when not calling the GUI from MATLAB.
`'TagsColumn'`	Name-Value	A non-negative integer specifying the column that corresponds to the tags currently assigned to the event code combination of that row of the csv file.
`'UseGui'`	Name-Value	If true (default), the *CTAGGER* GUI is displayed after initialization.

6. Tag format

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 specified in XML format, and *CTAGGER* provides a schema for validation. The map of tags to field values is specified in one of four possible formats: as a MATLAB structure array, as a JSON string, as a semi-colon/comma delimited string, or as a `tagMap` object. Most of the *CTAGGER* functions use the `tagMap` object representation, which provides methods to convert to and from the other representations.

6.1 XML tag hierarchy (HED)

The *CTAGGER* tools 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 Fig. 2.2. Internally, this hierarchy is represented as an XML string

Example 6.1: A snippet from XML representation of the tagging menu displayed in Fig. 2.2.

```
<?xmlversion="1.0" encoding="UTF-8"?>
<HED version="1.3">
<node default="true">
   <name>Time-Locked Event</name>
   <node>
      <name>Stimulus</name>
      <description>Input from outside world, exogenous</description>
      <node>
         <name>Visual</name>
         <node>
            <name>Fixation Point</name>
         </node>
         <node>
            <name>Shape</name>
               <node>
                  <name>Ellipse</name>
. . .
```

The XML hierarchy shown in Example 6.1 is from the `HEDSpecification1.3.xml` created by researchers at UCSD 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.

The *CTAGGER* tools work with any XML file that conforms to the `HEDSchema.xsd`, the default XML schema specification. The `HEDSchema.xsd` schema is quite general, and you can substitute any XML hierarchy that conforms to the schema or to build your own hierarchy from the ground up. You must take care in modifying the schema itself, as the community database and tagger GUIs assume 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`.

6.2 Tags are path strings

A tag is simply a path string from the root of the XML hierarchy to a specific node.

Example 6.2: The specification of a circle is the string:

```
'/Time-Locked Event/Stimulus/Visual/Shape/Ellipse/Circle'
```

Normally, when you add 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); it replaces the less specific tag. However, most *CTAGGER* functions take an optional `'PreservePrefix'` argument, which is `false` by default. If you set this argument to `true`, both tags are kept.

Example 6.3: When `'PreservePrefix'` argument is `true`, *CTAGGER* keeps all versions of the tags.

```
'/Time-Locked Event/Stimulus/Visual/Shape'
'/Time-Locked Event/Stimulus/Visual/Shape/Ellipse/Circle'
```

Regardless of the value of `'PreservePrefix'`, the *CTAGGER* database updates the counts for each node in the path from the most specific tag to the root only once.

6.3 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 6.4: 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, right, or center 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: 0.162048
   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.

The *CTAGGER* tools create 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'` will contain the associations between each of its two values (`'STIM'` and `'type'`) and the corresponding tags.

Example 6.5: The structure representation of the field map corresponding to Example 6.4 is:

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

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

Example 6.6: The structure representation of the tag map `'stimpos'` corresponding to the field map of Example 6.5:

```
fMap.map(1) =
   field: 'stimpos'
   values: [1x3struct]

fMap.map(1).values(1) =
   label: '1'
   description: 'Display of circle on left side of screen'
   tags: {'/Time-Locked Event/Stimulus/Visual/Shape/Ellipse/Circle' [1x54 char]}

fMap.map(1).values(2) =
   label: '2'
   description: 'Display of circle in the center of the screen'
   tags: ''

fMap.map(1).values(3) =
   label: '3'
   description: 'Display of circle on right side of screen'
   tags: ''
```

The tag map for the `'type'` field has the form:

```
fMap.map(2) =
   field: 'type'
   values: [1x2struct]
```

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 (the “field”). Tag maps can be represented by a MATLAB structure that has two fields (`.field`, and `.values`) at the top level. The `.values` field contains a structure with three fields (`.label`, `.description`, and `.tags`).

Internally, *CTAGGER* represents a tag map by a `tagMap` object. The `fieldMap` and `tagMap` objects allow the *CTAGGER* tools to be independent of a particular event and data representation.

6.4 Representing tags 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 considered to be lighter-weight than XML and is 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.

Example 6.7: JSON representation of the `.map` structure of Example 6.6:

```
{
   "field": "stimpos",
   "values":
       [
         {
            "label": "1",
            "description": "Display of circle on left side of screen ",
            "tags":
                [
                  "/Time-Locked
                        Event/Stimulus/Visual/Shape/Ellipse/Circle",
                  "/Time-Locked Event/Stimulus/Visual/Fixation Point"
                ]
         },
         {
            "label": "2",
            "description": "Display of circle in the center of screen ",
            "tags": ""
         },
         {
            "label": "3",
            "description": "Display of circle on right side of screen ",
            "tags": ""
         }
      ]
}
```

We use two *jsonlab* functions: `loadjson` and `savejson`. The `loadjson` function converts the JSON string to the MATLAB structure described in Section 6.3. The `savejson` function converts the MATLAB structure to JSON.

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

6.5 How tags are stored in a MATLAB structure

Tags can be stored in any dataset that is a MATLAB structure. *CTAGGER* 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 as shown in Example 6.8.

Example 6.8: Storing the field map structure of Example 6.6 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. In addition, for datasets that have events represented as a structure with fields, you can store the tags applicable to individual events.

Example 6.9: The tag information stored in the individual event of Example 6.4.

```
   EEG.event(1) =
      type: 'STIM'
      stimpos: 1
      latency: 162.048
      urevent: 1
      hedtags: ... direct mapped tags as a cell string
      usertags: {
                 '/Time-Locked Event/Stimulus/Visual/Shape/Ellipse/Circle',
                 '/Time-Locked Event/Stimulus/Visual/Fixation Point' ...}
```

The tags associated with a `'type'` value of `'STIM'` as well as a `'stimpos'` value of `'1'` are consolidated in `EEG.event(1).usertags` to allow data-mining. These tags are extracted from a field map that is also maintained to allow revision and remapping. Tags that come from automated annotation during data acquisition are stored in `.hedtags` and are not able to be remapped.

6.6 How tags are stored in a csv file

Tags can also be extracted from or stored in a csv file. The tags associated with a particular event type are concatenated and written to the tag file in column specified for tags. If the column is not specified, the tags are written to a column at the end of the data. If no description column is specified, a description is not extracted or rewritten.

Example 6.10: Tag a csv file using the `RewriteFile` input argument for `tagcsv`.

```
   fMap = tagcsv(filename, 'RewriteFile', './myoutfile');
```

If the `filename` argument in Example 6.10 represents a file with five columns, the values in those five columns are appended to form the event types (from the unique combinations). After the tagging process is complete, the `tagcsv` function writes a new file that contains the same rows as the original file, but the tags corresponding to the event combination for that row are written in an additional column. Additional parameters restrict the columns used for event types and rewriting.

6.7 The fieldMap object

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

Example 6.11: If you have several different maps, perhaps as a result of different people tagging the data or tagging for a different application, you can merge the field maps into a single `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 object using the default XML. The loop adds the individual group mappings to the object. You also 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 6.1: A summary of the arguments for the `fieldMap` constructor.

Name	**Type**	**Description**
`'Description'`	Name-Value	Description of this object.
`'PreservePrefix'`	Name-Value	If false (default), tags of 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.
`'XML'`	Name-Value	A string containing the HED tag hierarchy used to create this object.

Table 6.2: A summary of the public methods of the `fieldMap` class.

Method	**Description**
`addValues`	Include a structure or cell array of event values 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 the specified value of a specified 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.
`mergeDBXml`	Merge an XML string with the XML from the database.
`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 6.3: 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).

6.8 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 6.12: Representation of `fMap.map(1)` of Example 6.11 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 6.4: A summary of the arguments for the `tagMap` constructor.

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

Table 6.5: 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.
`getLabels`	Return the labels of keys associated with this object.
`getStruct`	Return this object as a structure.
`getText`	Return this object as semi-colon separated text.
`getTextValues`	Return the values of this object as semi-colon separated text.
`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 another `tagMap` object with this object.

Table 6.6: A summary of the public static methods of the `tagMap` class.

Method	Description
`createValue`	Create a structure to hold a value (label, description, tags).
`json2Mat`	Return a JSON representation of this object.
`json2Values`	Return a structure corresponding to a specified JSON string.
`Split`	Return the field name and value structure given text or JSON.
`value2Json`	Return a JSON representation of a value structure.
`value2Text`	Return a delimited text string representation of a value structure.
`values2Json`	Return a JSON string representation of a value structure array.
`values2Text`	Return a delimited text string representation of a value structure array.
`text2Mat`	Return a tag map structure corresponding to a delimited text string.
`text2Value`	Return a value structure corresponding to a delimited text string.
`text2Values`	Return a value structure array given a cell array of delimited text strings.
`validateValue`	Validate that a structure corresponds to a value.

6.9 The csvMap object

The `csvMap` class encapsulates the values in a csv file. These values correspond to the event types of the data as well as associated tags and a description if specified.

MATLAB Syntax

```
   cMap = csvMap(filename)
   cMap = csvMap(filename, varargin)
```

Table 6.7: A summary of the arguments for the `csvMap` constructor.

Name	Type	Description
`'FileName'`	Required	The full path name of the csv file.
`'Delimiter'`	Name-Value	Delimiter between tokens in the key.
`'DescriptionColumn'`	Name-Value	Column number of description column.
`'EventColumns'`	Name-Value	Numbers of the columns of event key labels.
`'TagsColumn'`	Name-Value	Column number of tag column.

Table 6.8: A summary of the public methods of the `csvMap` class.

Method	Description
`addEvent`	Add event row to values and returns position or 0 if not added.
`getEvents`	Return a structure array of event structures.
`getHeader`	Return a cellstr array with the tokens in first line of file.
`getLabels`	Return the unique labels for this map.
`getType`	Return a string containing the type names as a key.
`getValue`	Return the value structure corresponding to specified label.
`getValues`	Return the cell array of values from the original file.
`updateDescriptionValues`	Update the values cell array based on the description in a specified tag map.
`updateTagValues`	Update the values cell array based on the tags in a specified tag map.
`writeTags`	Write the tags in csv format given a specified tag map.

Table 6.9: A summary of the public static methods of the `csvMap` class.

Method	Description
`getkey`	Return a key given the column(s) of value.
`getval`	Return the value of a particular column.

6.10 Representing a tag map as a delimited string

The delimited string representation is not as flexible as the other representations. The format is a semicolon-separated list consisting of the field name followed by a representation of each value-tag map. The label, description, and tags for each value are separated by commas. Only the tag maps and not the field maps can be represented by delimited strings.

Example 6.13: Text representation of the tag map of Example 6.4.

```
'stimpos;1,STIM,Display of circle on left side of screen,/Time-Locked
 Event/Stimulus/Visual/Shape/Ellipse/Circle, /Time-Locked
 Event/Stimulus/Visual/Fixation Point;2,STIM,Display of circle in the center of
 screen,;3,STIM,Display of circle on the right side of screen,'
```

7. Saving tags in the data set (the `writetags` function)

The *CTAGGER* `writetags` function is used by other functions to write the tag information to dataset represented by a MATLAB structure. *CTAGGER* 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, *CTAGGER* assumes that the events to be tagged are in stored in the `.event` structure array and 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 7.1: Write the tag encapsulated by the `fieldMap` object `fMap` into the data structure `x`.

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

By default *CTAGGER* 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.

Example 7.2: Only write tag information for individual events into the data structure `x`.

```
x = writetags(x, fMap, 'RewriteOption', 'Individual');
```

MATLAB Syntax

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

Table 7.1: Summary of the arguments for 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), 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.
`'RewriteOption'`	Name-Value	A string indicating how tag information should be written to the datasets. The options are `'Both'`, `'Individual'`, `'None'`, `'Summary'`. See the section on rewriting tags to datasets for additional explanation.

Table 7.2: Different tag rewrite options

Value	Description
`'Both'`	Summary tag information is written to `.etc.tags` and tags for individual events are written into `.event(i).usertags`.
`'Individual'`	Tags for individual events are written into `.event(i).usertags`.
`'None'`	No tags are written.
`'Summary'`	Summary tag information is written to `.etc.tags`.

8. Running the regression tests

CTAGGER uses the XUNIT unit testing framework for its regression tests. For tests that require user input, the instructions appear in caps in the command window. Some test data is available in the `tests/testData` subdirectory that comes with the distribution. However, to run the tests on tagging directories and studies require a larger dataset that is available separately from the CTAGGER distribution. Download the test data archive (`CtaggerTestArchive.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.

9. Status and availability

The base *CTAGGER* GUI is currently available and undergoing user testing. The database is currently being developed and tested.

10. Acknowledgments

The authors acknowledge helpful conversations with Nima Bigdely Shamlo, Christian Kothe, 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, and Tony Johnson of DCS Corporation. 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.

11. 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].