**CTAGGER: User Guide (Standalone)**

Contents

[1. Getting Started with *CTAGGER* 3](#_Toc417225103)

[1.1 Overview 3](#_Toc417225104)

[1.2 Requirements 3](#_Toc417225105)

[2. Data formats 3](#_Toc417225106)

[2.1 Experimental events 3](#_Toc417225107)

[2.2 Hierarchical event descriptors (HED) 4](#_Toc417225108)

[2.3 Representing events or event-codes as tab-delimited text 4](#_Toc417225109)

[2.4 Representing events in XML 5](#_Toc417225110)

[3. Using *CTAGGER* 6](#_Toc417225111)

[3.1 Running *CTAGGER* 6](#_Toc417225112)

[3.2 The *exit* button 6](#_Toc417225113)

[3.3 Using the *load* and *save* buttons 7](#_Toc417225114)

[3.3 Tab-delimited text options 7](#_Toc417225115)

[3.4 The event view 8](#_Toc417225116)

[3.5 *CTAGGER* selections and the *deselect all* button 9](#_Toc417225117)

[3.6 The *add group* button 9](#_Toc417225118)

[3.7 Tagging a tag group 9](#_Toc417225119)

[3.8 Missing tags 9](#_Toc417225120)

[3.9 Tagging with required and recommended tags 10](#_Toc417225121)

[3.10 Zooming the display 10](#_Toc417225122)

[3.11 Using *undo/redo* buttons 10](#_Toc417225123)

[3.12 Adding an event 11](#_Toc417225124)

[3.13 Removing a tag from an event or tag group 11](#_Toc417225125)

[3.14 The tag view and the *collapse* and *expand* buttons 11](#_Toc417225126)

[3.15 Adding a tag to the hierarchy 12](#_Toc417225127)

[3.16 Tag attributes 13](#_Toc417225128)

[3.17 Searching tags 14](#_Toc417225129)

[4. Sample data and examples 15](#_Toc417225130)

[5. Status and availability 15](#_Toc417225131)

[6. Acknowledgments 15](#_Toc417225132)

[7. References 16](#_Toc417225133)

# 1. Getting Started with *CTAGGER*

## 1.1 Overview

*CTAGGER* is a program designed to help users annotate events or other data elements using a predefined, but extensible, hierarchically structured annotation language. Although *CTAGGER* can be used in very general contexts, this document describes its use specifically for tagging events in EEG data. The input to the system consists of two parts: a list of items to be annotated or tagged and an annotation hierarchy. In the case of EEG, users annotate the events that occur during an EEG experiment using the HED 2.01 hierarchical event description language as the vocabulary. The *CTAGGER* allows users to annotate their events quickly and accurately by selecting tags from the HED hierarchy.

## 1.2 Requirements

*CTAGGER* has been tested on systems running either 64-bit Windows 7 or Ubuntu. The only requirement to run *CTAGGER* is a Java JRE of version 1.6 or later.

# 2. Data formats

## 2.1 Experimental events

A typical EEG analysis proceeds by analyzing brain signals relative to the events that occur during the experiment. In the past, analyses have focused on single narrowly defined experiments, and researchers have devised their own schemes for encoding the relevant events. However, as impetus to share data and perform analysis across experiments, the need for a common vocabulary for identifying events has become acute.

The *CTAGGER* starts with experiment-specific event representations and helps researchers place their experiments in a larger context by associating their private event codes with a common community vocabulary. This common vocabulary consists of tags, which 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. Users may group event tags with one level of parentheses to make the annotation clearer. Figure 2.1 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.

**Figure 2.1:** Tag path string representation

/Event/Category/Experimental stimulus,

(/Item/2D shape/Ellipse/Circle, /Attribute/Visual/Color/Red, /Location/Screen/Center)

*CTAGGER* allows two general strategies for tagging events: by code and by latency. Tagging by code produces a table associating an event code with a list of tags. In controlled experiments, researchers determine the events as part of the experimental design and usually represent each event type by a code. Events associated with each event code occur many times during the course of an experiment. Since the event code determines the tags for any of the events associated with the code, researchers only need to tag the event code once. They can then use the output of *CTAGGER* as a lookup table for tagging events by their codes. The MATLAB version of *CTAGGER* has support for doing this in EEGLAB EEG structures. Researchers can tag multiple tables of event types (codes) and sub-codes to support more complex tag annotation.

Tagging by latency produces a table associating an event latency with a list of tags. In more complex or real world experiments, researchers do not have a fixed set of event codes. Rather, events are identified on the fly or by post processing of experimental output such as video, audio, and output of simulators or other instrumentation. From the viewpoint of the *CTAGGER* standalone software, there is little difference between tagging by code and tagging by latency. The code or the latency merely serves to identify the event/code to tag. Both *CTAGGER* refers to both as “events”.

## 2.2 Hierarchical event descriptors (HED)

The *CTAGGER* assumes that rather than inventing tags at random, you will have a menu of predefined suggested tags presented in hierarchical form. For EEG applications, we will be using the HED 2.01 hierarchy for event annotation. This hierarchy is stored in XML format as shown in Figure 2.2. The *CTAGGER* provides a user-friendly display of this information to make it easier for users to annotate their data.

**Example 2.2:** A snippet from XML representation of the HED 2.01 hierarchy

<?xml version="1.0" encoding="utf-8"?>

<HED version="2.01">

<node>

<name>Event</name>

<node position="1" requireChild="true" required="true">

<name>Category</name>

<description> This is meant to designate the reason this

event was recorded</description>

...

</node>

...

Researchers at University of California San Diego (UCSD), the University of Texas at San Antonio (UTSA), and Syntrogi Corp created the HED language 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.

## 2.3 Representing events or event-codes 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. *CTAGGER* supports loading of events and their annotations in a tab-delimited text format for this purpose. Each row or line of the file represents an event, with the event code or event latency in a particular column and tags optionally in other columns. *CTAGGER* prompts for the number of header lines, the event code columns, and the tag columns. *CTAGGER* ignores the header lines and uses the event code columns to identify the events. These columns could contain unique codes, event latencies, or both.

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. Figure 2.3 shows an example with three events (the lines are 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.

**Figure 2.3:** 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)

## 2.4 Representing events in XML

The XML format allows users to save tagged events and the corresponding tag hierarchy into a single XML file. Figure 2.4 shows the beginning of an XML file with a single tagged event, which contains both event-level tags and a tag group.

**Figure 2.4:** A snippet of code representing events in XML

<?xml version="1.0" encoding="UTF-8"? >

<TaggerData>

<eventSet>

<event>

<code>1111</code>

<tag>/Event/Category/ExperimentalStimulus</tag>

<tag>/Event/Label/LeftPerturbOnset</tag>

<tagGroup>

<tag>/Item/Object/Vehicle/Car</tag>

<tag>/Attribute/Object control/Perturb</tag>

<tag>/Attribute/Direction/Left</tag>

</tagGroup>

</event>

</eventSet>

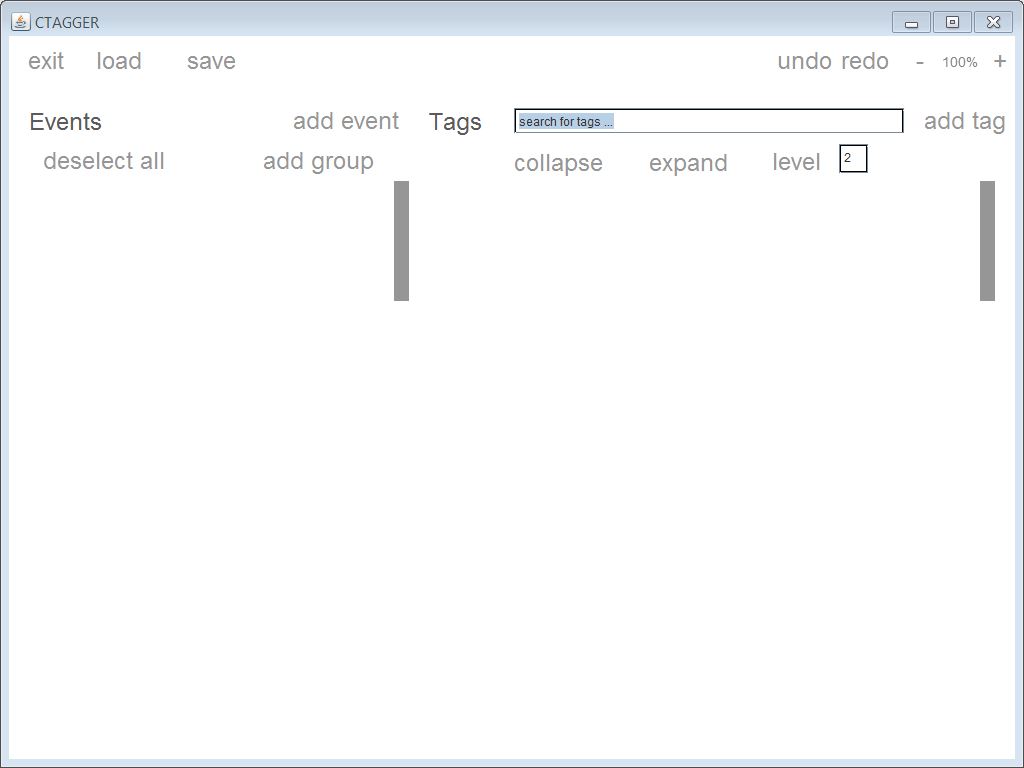
...

# 3. Using *CTAGGER*

## 3.1 Running *CTAGGER*

To run *CTAGGER* simply double click on the CTagger.jar file. Figure 3.1 shows an example of *CTAGGER* after it loads.

**Figure 3.1:** Example of *CTAGGER* interface after it has loaded.



## 3.2 The *exit* button

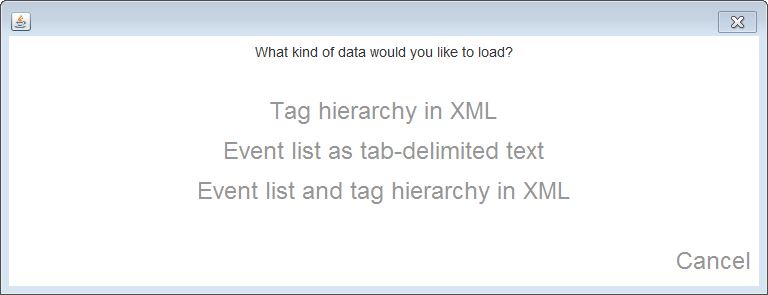
The *exit* button appears in the upper left hand corner, above the event view. This button simply closes the *CTAGGER*. If you have loaded events or a tag hierarchy, *CTAGGER* will prompt you about whether or not to save the current state.

## 3.3 Using the *load* and *save* buttons

You can use the *load* button at any time to load a tag hierarchy in XML format, a tab-delimited file of events, or a file that combines both in XML format. If events or tags have already been loaded, loading again simply replaces the old values, discarding any changes. Press the *save* button to save the current state of the *CTAGGER*. Note that when you choose to save the file in tab-separated format, *CTAGGER* just saves one column with the event code or latency and a second column with the complete list of tags for the event or code.

As Figure 3.2 shows, *CTAGGER* presents the following three options when both the *load* and *save* buttons are pressed: *Tag hierarchy in XML*, *Event list as tab-delimited text*, and *Event list and tag hierarchy in XML*. If you choose the *Tag hierarchy in XML* option, *CTAGGER* only saves the tag hierarchy displayed in the right panel. Most users will not need to modify the hierarchy unless the tag hierarchy does not cover the events that occur in their experiment. If you choose the *Event list as tab-delimited text* option, *CTAGGER* saves the table of tagged events. If you choose the *Event list and tag hierarchy in XML* option is chosen, *CTAGGER* saves both the events and tags in a single file in XML format. This format is useful for downstream tools.

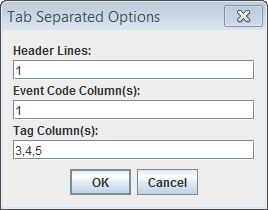
**Figure 3.2:** Example of the save and load options



## 3.3 Tab-delimited text options

If you load event data using tab-delimited text option*, CTAGGER* will prompt you to specify the layout of your file as shown in Figure 3.3. The header lines specify how many lines at the beginning of the file *CTAGGER* should ignore before processing events. *CTAGGER* assumes the file has tab-delimited columns. The event code columns contain the event codes used to identify the event item. Depending on the tagging strategy, these are usually either the researchers’ internal event codes or the event latencies. The tag columns contain lists of tags, which are path strings selected from the tag hierarchy. Specify a zero (0) if your file does not have any columns with tags. To specify multiple columns separate each column with a comma (i.e., 3,4,5 for columns three, four, and five).

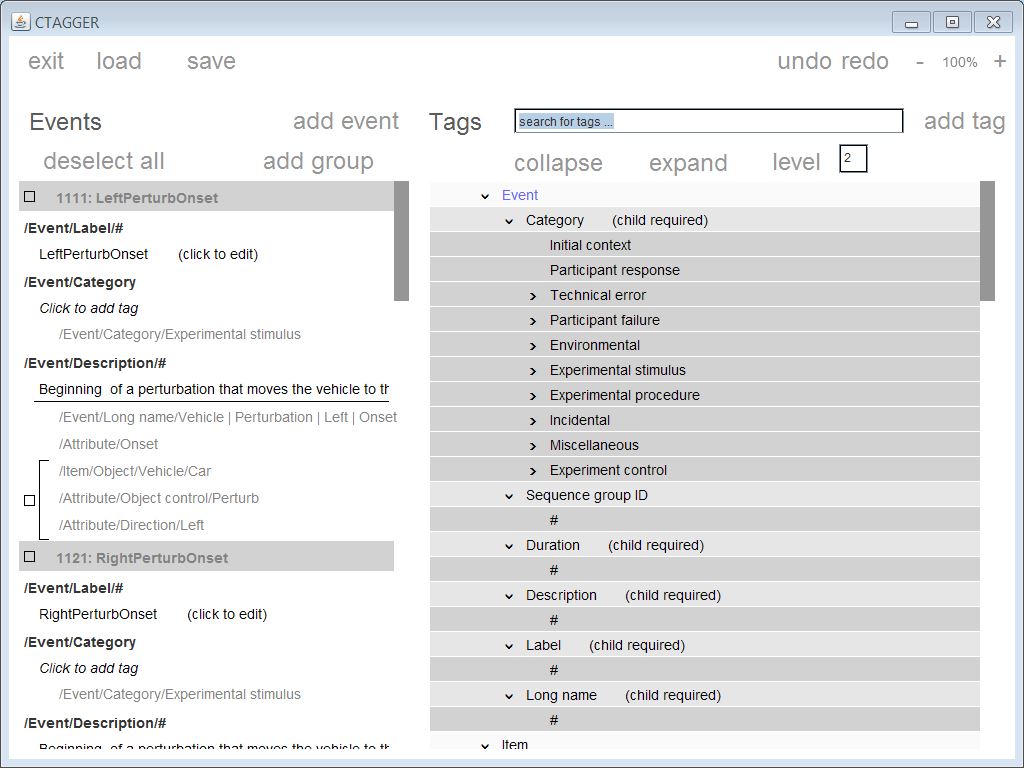
**Figure 3.3:** Example of the tab-delimited options



## 3.4 The event view

*CTAGGER* displays each event in the left panel as an identifier (code and/or label) followed by the tags associated with the event. Figure 3.4 shows and example of this display. The first tags listed are the required and recommended tags, as specified by tag attributes in the HED file. These tags include the category, label, and description. If any of these tags do not meet certain requirements such as a label not having an actual value, *CTAGGER* displays the tag in red if required and orange if recommended. You can hide the section showing required and recommended tags can by right clicking the event header and choosing *hide required/recommended*. Below this section are event level tags and tag groups associated with the event. *CTAGGER* shows tag groups using a square bracket to the left of the group.

**Figure 3.4:** Example of loaded events and tags in the *CTAGGER* display.



## 3.5 *CTAGGER* selections and the *deselect all* button

Each event and each tag group within an event has check box indicating selection. When you click on a tag in the tag hierarchy on the right, *CTAGGER* automatically adds it to all selected tags. To avoid spurious tagging, you can press the *deselect all* button on the upper left side to make sure that *CTAGGER* does not have any items selected. If you click on an event header area outside of the checkbox, it will become the only one selected (all others will be deselected).

## 3.6 The *add group* button

The *add group* button inserts a single tag group in all selected events. *CTAGGER* will not do anything if no events are selected or only tag groups are selected.

## 3.7 Tagging a tag group

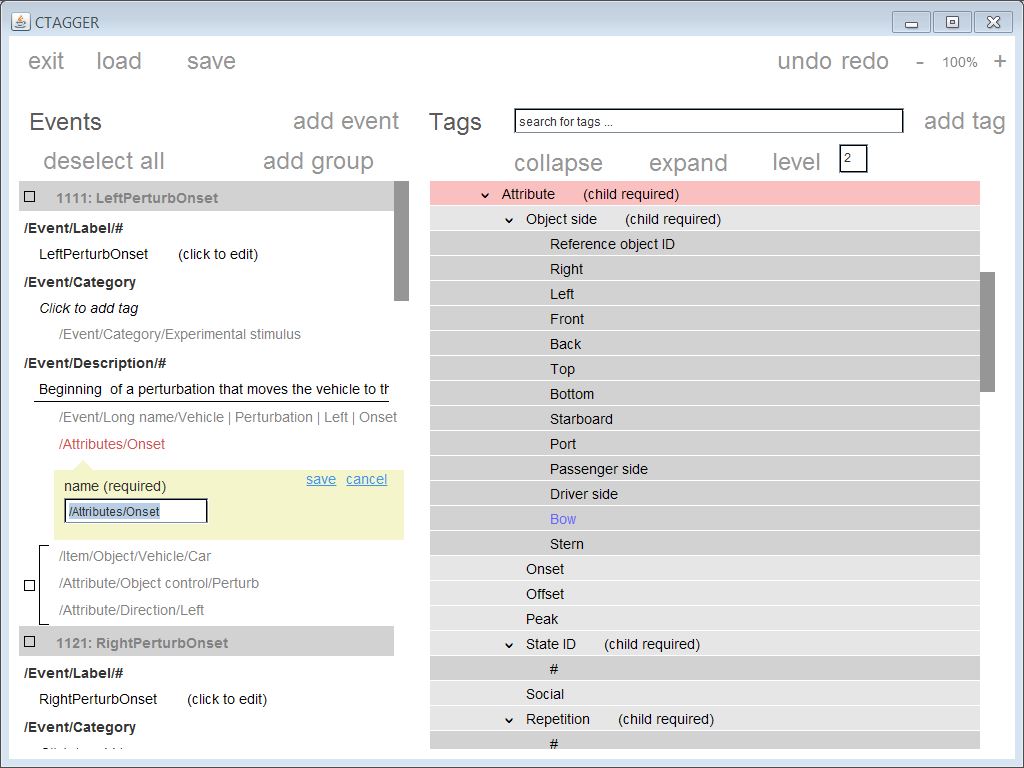
Tags are associated with events but can also be grouped together to form a tag group. Right-click the event header and choose the *add group* option from the menu to add a new tag group to an event. An empty group will appear below and will now be available to tag. To associate or dissociate a tag with a selected group, click on a tag in the hierarchy on the right. This has the effect of toggling the association of the tag and the selected groups or events. If the tag is currently missing from any of the selected groups, *CTAGGER* will add it to those groups. If the tag is present in all of the selected groups, *CTAGGER* will remove it. Thus, you can make sure that the tag is in all selections by clicking the tag in the hierarchy either two or three times.

Certain conflicts may prevent a tag from being added. Some tags are marked with the “unique” attribute, meaning that only one of this tag or its descendants may be used within an event or group. If a conflict exists, the *CTAGGER* will present a dialog explaining the conflict and give any options available. For example, there is an option to replace a general tag with a more specific tag. Other types of conflicts may have to be resolved by manually removing conflicting tags before adding others.

## 3.8 Missing tags

*CTAGGER* highlights missing tags in red under its respective event or group. When a tag is missing, you have the option to right click on it and select *edit* to change the tag path. *CTAGGER* allows editing because there may be some typos when specifying the tags. A user can only change the existing tag path to a tag path that exists in the hierarchy. Figure 3.5 shows an example.

**Figure 3.5:** Example of a tag path being edited



## 3.9 Tagging with required and recommended tags

Required and recommended tags can be added to an event in the same ways as other tags, though you can only add these tags at the event level (above the horizontal separator) rather than to a tag group. You can also add or remove by editing the tag in the hierarchy and by selecting or deselecting the required or recommended attribute.

## 3.10 Zooming the display

The *–* *100%* *+* buttons control the size of the display and allow you to reduce (*–*) or expand (*+*) the view. *CTAGGER* displays the current percentage of zoom between the two buttons.

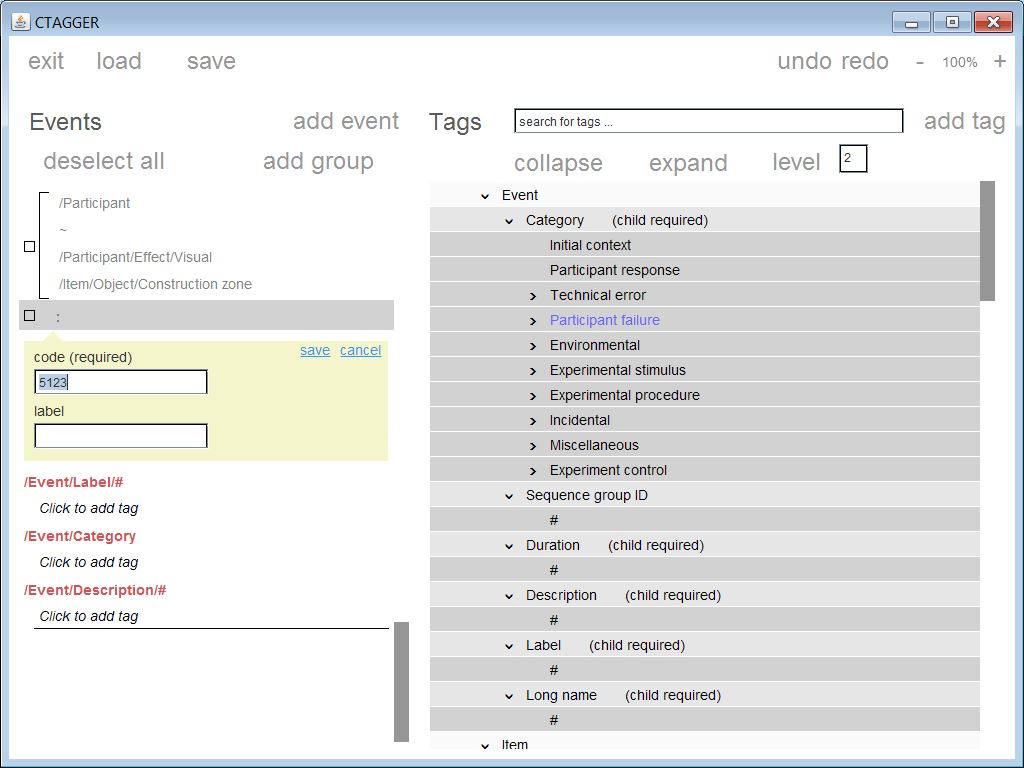
## 3.11 Using *undo/redo* buttons

The *undo* and *redo* buttons on the top upper right allow you to undo or redo tagging operations. You can reverse or reapply most actions (e.g., associations, adding and deleting tags, etc.) using these options. When the mouse is hovered over the *undo* or *redo* button, a message appears to show what kind of action the button will perform next if clicked.

## 3.12 Adding an event

To add an event, click *add event* on the left side, above the event view. *CTAGGER* opens a newly added event for editing as shown in Figure 3.6. After you have specified the required information, click the local *save* button if you wish to keep the event or the local *cancel* button to remove it. Note, saving the event in the add event panel only adds it to your current tag document, but does not save it to disk. To save your tagging to disk, you need to use the *save* button above the event panel.

**Figure 3.6:** Example of adding an event in *CTAGGER*.



## 3.13 Removing a tag from an event or tag group

In addition to the toggling method described above, you can remove a tag from an event or tag group by right clicking on the tag in the event view and choosing the *remove* option from the menu.

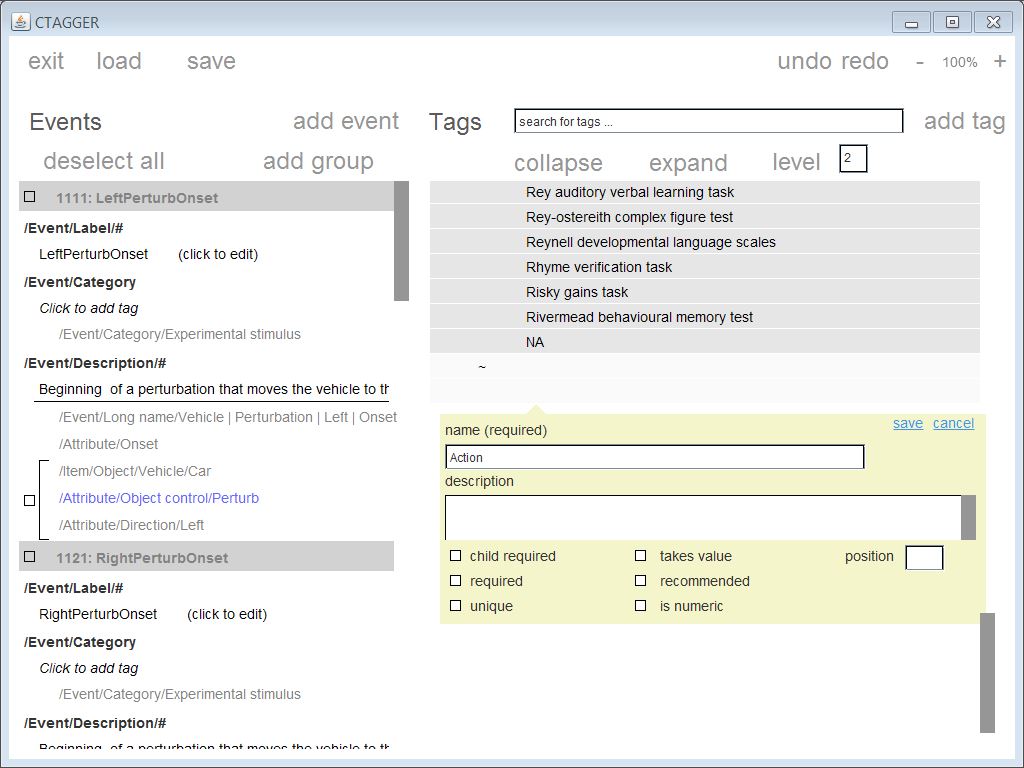
## 3.14 The tag view and the *collapse* and *expand* buttons

*CTAGGER* displays the tag hierarchy on the right panel. You can expand or collapse levels by clicking on the arrows next to certain tag names. *CTAGGER* highlights tags at the same level in the same color. Darker shades correspond to deeper levels in the tag hierarchy. Use the *collapse* button to collapse all levels of the tag hierarchy. Use the *expand* button to expand completely expand the hierarchy. Enter a level to expand to a certain level. Level 0 corresponds to the top level forest.

## 3.15 Adding a tag to the hierarchy

To add a tag underneath an existing tag, right-click on the tag in the hierarchy that you wish to add under (i.e. the parent of the new tag), and choose the *add tag* option. *CTAGGER* will display a new tag in the hierarchy at the desired level open for editing as shown in Figure 3.7. You can enter the name, which is required as well as a description and other attributes. After inputting the information click local *save* to apply these changes. The local save only adds the tag to the hierarchy currently in memory. You must do the top-level save to save a modified hierarchy to disk. To add a top-level category in the hierarchy, click the *add tag* button above the hierarchy.

**Figure 3.7:** Example of a new tag being added to the hierarchy.



## 3.16 Tag attributes

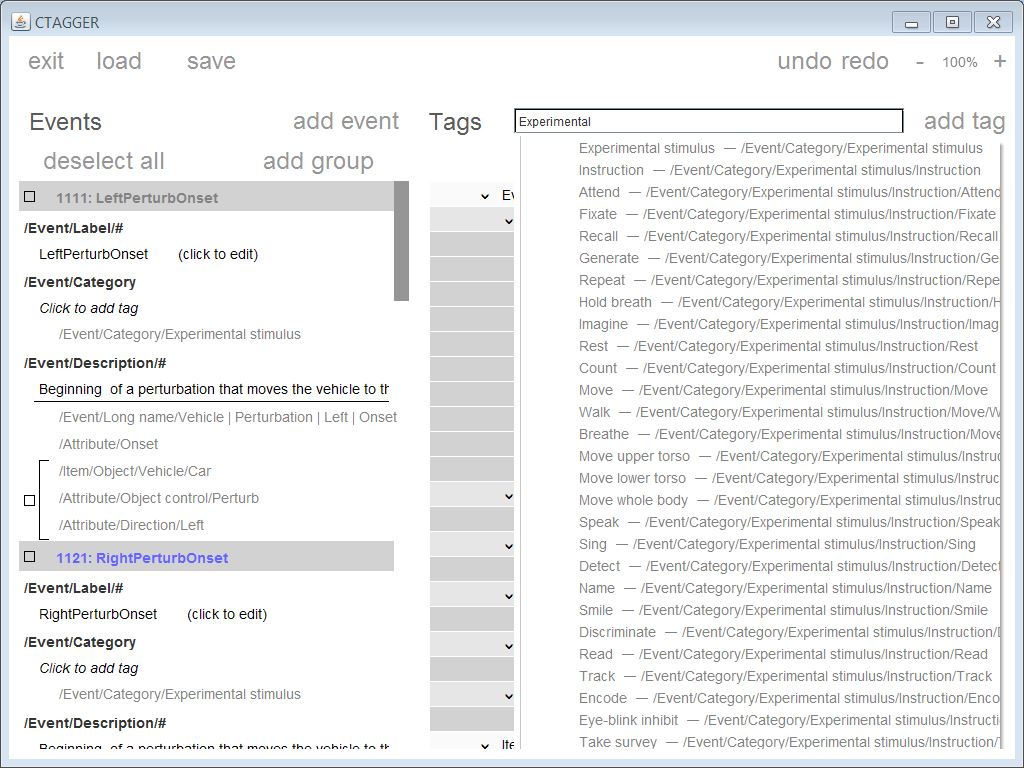
All tags have basic fields such as name and description that help users identify and understand the tag. However, certain attributes affect the way that *CTAGGER* can use the tag. There are currently seven attributes that tags can have:

* requireChild (child required) - One of its descendants must be chosen to tag an event.
* takesValue (takes value) - The tag name is the "#" character. This character will be replaced with the user's input. The input is a string.
* isNumeric (is numeric) - The tag name is the "#" character. This character will be replaced with the user's input. The input is numeric.
* required - Shown at the top of the events view in the order given by the position attribute.
* recommended - Shown below the required tags in the events view in the order given by the position attribute.
* position - Used to specify the order of the required and recommended tags (compared separately from each other). *CTAGGER* expects an integer and the order can start at 0 or 1; it just checks for comparison. Required or recommended tags without this attribute (or with negative position) will be shown after the others.
* unique - Only one of this tag or its descendants can be used within a single tag group or event.

## 3.17 Searching tags

You can search for tags in *CTAGGER* using the search bar and typing in a particular word or phrase as shown in Figure 3.8. Options for tags matching the search term will appear below. Click on one of these options to locate the tag in the hierarchy. The GUI will scroll to the tag and highlight it. In addition, when you click on a tag in the event view on the left, it will also scroll to and highlight the tag on the right. *CTAGGER* highlights tags that take values in yellow and other tags in blue. If you click on a tag that does not exist in the hierarchy, *CTAGGER* will scroll to the nearest match found and highlight it in red. If there is no match anywhere in the hierarchy, *CTAGGER* displays a message at the top of the screen.

**Figure 3.8:** Example of the tag search



# 

# 4. Sample data and examples

The CTAGGER distribution includes a data directory containing the following six (6) files:

* BCI Data Specification v15 Tagger Data.xml (*Event list and tag hierarchy in XML*)
  + The BCI Data Specification v15 Tagger Data.xml contains the events following by the tags in XML format.
* BCI Data Specification v15 TSV.txt (*Event list as tab-delimited text*)
  + When loading the BCI Data Specification v15 TSV.txt and specifying the tab-delimited options use 1 for the header lines, 1 for the event code column, and 3,4,5,6 for the tag column.
* HED2.01.xml (*Tag hierarchy in XML*)
  + The HED2.01.xml is the latest version of the hierarchical event description language used for tagging.
* LSIE\_05\_allevents\_indoor.txt (*Event list as tab-delimited text*)
  + For the LSIE\_05\_allevents\_indoor.txt tab-delimited options use 7 for the header lines, 5 for the event code column (*EEG Time*), and 10 for the tag column.
* LSIE\_05\_allevents\_outdoor.txt (*Event list as tab-delimited text*)
  + For the LSIE\_05\_allevents\_outdoor.txt tab-delimited options use 6 for the header lines, 6 for the event code column (*EEG Time*), and 11 for the tag column (*Hed 2.01*).
* TX16 HED Tags.txt (*Event list as tab-delimited text*)
  + For the TX16 HED Tags.txt tab-delimited options use 1 for the header lines, 1 for the event code column (*Event Number*), and 7 for the tag column (*Combined*).

# 5. Status and availability

The base *CTAGGER* is currently available and undergoing user testing. The database is currently being developed and tested. The latest version of *CTAGGER* can be download [here](https://github.com/VisLab/CTagger-Java).

# 6. Acknowledgments

*CTAGGER* development has been occurring in the laboratory of Kay Robbins of the University of Texas at San Antonio. The developers wish to acknowledge helpful conversations with Christian Kothe, Nima Bigdely Shamlo, Alejandro Ojeda formlerly of the University of California at San Diego and now of Syntrogi. Arno Delorme, and Scott Makeig 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 Grant Hanada of the University of Michigan. The original CTAGGER interface was developed by Thomas Rognon with contributions from Jeremy Cockfield, Lauren Jett, Rebecca Strautman, and Cassidy Matousek. 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.

# 7. 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].