**EEG Tagger – defining a common language for event-rich environments**

**Thomas Rognon, Lauren Jett, Rebecca Strautman and Kay Robbins UTSA**

## 1. Introduction

Community tagging, also known folksonomy or collaborative tagging, refers to systems in which a community of users of information assign tags to data elements in order to 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.

In scientific experiments using *event-rich environments*, the need for a common method of annotation is critical for data sharing and the application of data mining and machine learning techniques across multiple studies. An *event* in this context refers to a labeled occurrence that happens at a particular point in time. Events have a type or label, which may be common to multiple events or occurrences. In some representations, events have both starting and ending times. Other representations define separate events at the starting and ending times of incidents that have finite duration. An example of an event-rich scientific environment is real-time monitoring to capture brain and body data during active behavior using multi-channel EEG, eye tracking, and motion capture.

Traditional EEG research uses externally-generated time-stamped event codes to indicate presentation of stimulus and user response. The events play a central role, since many analyses begin by epoching the data around these events and looking for differences by contrasting behavior in epochs associated with different events. Even in this limited event context, there is no standardization of annotation. One researcher may use the code 302 to represent a button press in response to a light stimulus and 304 to represent a button press in response to a sound. Another researcher may use completely different codes or even descriptive names. In order to perform analysis across datasets, the analysis must map event codes across the experiments. This process usually requires detailed contextual information that is often not available within the data itself and may not be available at all.

The events that typically represent the experimental stimuli and subject responses are a small subset of the possible information that might be encoded in events. Events can also be generated by time-synchronized output from simulators and sensing devices such as eye-trackers and motion capture. Events may be defined to encode a variety of environmental factors. For example, an experiment in which the subject hears audio communication from a variety of sources as a distractor in performing a visual task, will usually record a synchronized audio stream for additional processing. Processing may convert this audio stream into an event stream with events such as “Priority command audio starts” or “Music stops”. The type of events extracted will likely depend on the analysis performed and many event overlays are possible from a source such as an audio stream. Signal features may be extracted as events, as may the output of classification algorithms. A single experiment could have many possible event overlays corresponding to different analyses. Without a common labeling scheme across experiments and analyses, reuse, sharing and large scale application of analyses is extremely limited. The next section describes an approach to imposing order on event nomenclature using structured tagging in a community setting.

## 2. Structured Tagging using hierarchies

Structured tagging is a way of imposing order and a common vocabulary on event nomenclature across experiments. The idea is for experimenters to assign descriptive tags to each unique type of event. Analysts can then use the tags to extract similar events across experiments. The scheme consists of three elements:

* A predefined hierarchy of tags (HED xml specification)
* A flexible interface that allows users to quickly associate tags with event types and to add elements to the hierarchy as needed (cTagger)
* A database that consolidates hierarchies across experiments and laboratories, allowing users to select frequently-used tags where appropriate and to benefit from nomenclature development of other scientists (cTaggerDB)

Each of these elements is described more completely in the following subsections.

### 2.1 Tags as paths and the HED XML specification

In developing a vocabulary or nomenclature to describe events there is a tradeoff between structure and expressiveness. Consider a simple visual stimulus consisting of a red circle. One researcher may label this type of event as “stimulus” or “circle” or “visual” or some combination. The circle may be thought of as a special case of an ellipse or more generally a shape. Tags are organized into hierarchies and specified as paths to capture the different types of labeling. For example, the user may label this type of event simply as: /Stimulus/Visual or more specifically as /Stimulus/Visual/Shape/Ellipse/Circle. The later tag is more specific and supersedes the former. The user may also want to specify the color, for example as /Stimulus/Visual/Uniform Color/Red.

Nima Bigdely-Shamlo et al. (2012) of UCSD created the HED xml specification to assist experimenters in labeling their data. The initial specification of HED provides a starting point for investigators who wish to label events, particularly those in EEG studies, with informative tags that convey common meanings.

### 2.2 The cTagger user interface to facilitate tagging

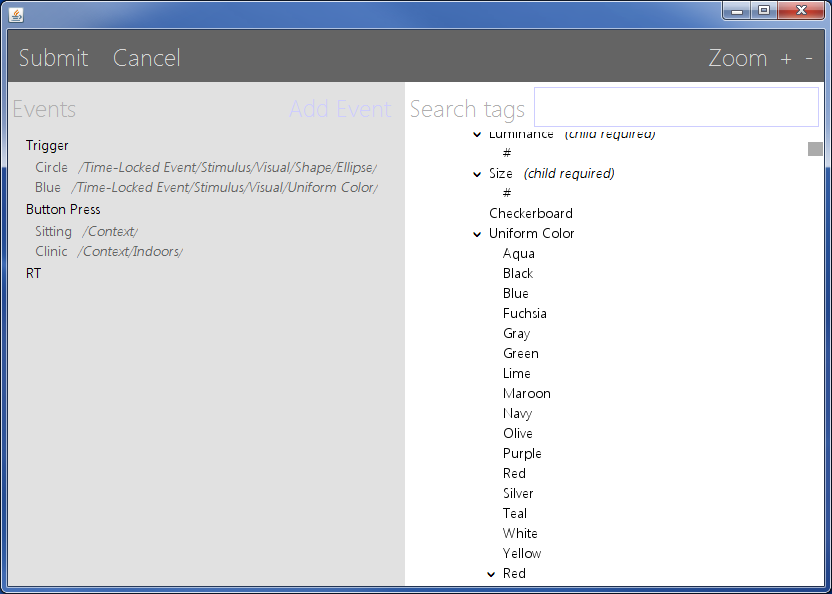
To facilitate tagging, Thomas Rognon and Kay Robbins of UTSA have developed a user interface called cTagger, which can be called as a standalone program or from MATLAB to facilitate tagging. A screenshot of cTagger is shown in Figure 1. The GUI left panel is initialized with a list of event types and their associated tags thus far. The GUI right panel is initialized with the hierarchy of potential tags (e.g., the HED hierarchy described above). Users can add event types on the left by pressing Add Event. To tag event types with elements of the hierarchy, the user can simply drag and drop tag paths from the right panel to the particular event. Alternatively, the user can select a particular event type on the left and then successively click or unclick designed tags on the right.

Figure 1: cTagger GUI

To allow flexibility and expansion of the vocabulary, the user can add tags to any point in the hierarchy by right clicking in the right pane at the point of addition. When the user clicks on submit, the GUI returns with the tagged event types and the modified hierarchy.

Usually, a group of experiments with common event types will form a study, and an experimenter will only have to label the event types for one experiment and then apply to all. The user can start with a pretagged list of event types as input to the GUI and modify or make additions to the tagging.

### 2.3 Community tagging to support the development of a common vocabulary

The UTSA group is also developing a backend database for cTagger, called cTaggerDB. When the user presses submit, the tag counts and updated hierarchy are merged with a global hierarchy held in a database. In the community version, the database keeps track of additions in the hierarchy so users can take advantage of vocabulary developed by others. The database also keeps track of the number of times each tag is used and the GUI displays these counts in parentheses. The counts provide a suggestion to experimenters about which tags are likely to be commonly expected to facilitate data mining. In the community version, the GUI also supports user annotation of the hierarchy. These annotations or comments are not stored in the hierarchy, but are stored in the database. Users of the GUI can click on an element to view the annotations when using the GUI with a database.

## 3. Tag format

Community tagging is structured and hence requires two items: a tag hierarchy and a map of tags to events. The tag hierarchy is specified in XML format and a schema is provided for validation. The map of tags to events is specified in the four possible formats: as a JSON string, as a MATLAB structure array, as a delimited string, or as an eventTags object. Most of the cTagger functions use the eventTags object representation, which can convert to and from the other representations.

### 3.1 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. We use the jsonlab MATLAB library[2], which translates between JSON strings and MATLAB structures.

An example of the JSON format is shown below. At the top level there are two JSON elements, "hedXML" and "events". The "hedXML" element should either be empty or a valid XML string, although in the example below we show it as "abc" for brevity. The "events" element can be empty, an event element or an array of event elements. An event always has "code", "label", and "description" elements that could be strings or empty. The "tags" element could be empty, a string, or an array of strings:

{

"hedXML": "abc",

"events": [

{

"code": "1",

"label": "RT",

"description": "RT",

"tags": [

"/Time-Locked

Event/Stimulus/Visual/Shape/Ellipse/Circle",

"/Time-Locked Event/Stimulus/Visual/Fixation Point"

]

},

{

"code": "2",

"label": "Trigger",

"description": "Trigger",

"tags": ""

}

]

}

We use two jsonlab functions: loadjson function converts the JSON string to the MATLAB structure described in Section 3.2. The savejson converts the MATLAB structure to JSON.

jStruct = savejson('', jString);

jString = loadjson(jStruct);

### 3.2 Representing tags as a MATLAB structure

As shown in the example below structure representation has t fields at the top level: hedXML and events. The field either is empty or contains a structure array of events.

jStruct =

hedXML: 'abc'

events: [1x2 struct]

where:

jStruct.events(1) =

code: '1'

label: 'RT'

description: 'RT'

tags: {'/Time-Locked Event/Stimulus/Visual/Shape/Ellipse/Circle' [1x54 char]}

### 3.3 Representing event tags as a delimited string

The delimited string representation is not as flexible as the other three representations. The format is a single string which is the concatenation of the HED XML string followed by a representation of each event. These items are separated by semicolons:

'abc;1,RT,RT,/Time-Locked Event/Stimulus/Visual/Shape/Ellipse

/Circle, /Time-Locked Event/Stimulus/Visual/Fixation Point;2,

trigger,trigger,'

Each individual event is encoded as a comma-separated string: code, label, description, tag1, tag2, etc.. The code and label fields must not be empty. Because semicolons are used as item delimiters, the semicolon character should not be used in the HED XML or within the events.

### 3.4 Representing event tags as an eventTags object

The eventTags class is designed to provide a common format for holding the tagging information. It has methods for translated to and from the other formats and for merging event objects.

To create an eventTags object from a JSON string use:

myEvents1 = eventTags(inString1);

To create an eventTags object from a text string use:

myEvents2 = eventTags(inString2, 'UseJson', false);

Both of these versions assume that equality of events will be determined by their code values. To use the label value instead, you must specify the type of comparison to be used:

myEvents3 = eventTags(inString3, 'Match', 'label');

The value of the match argument can be 'code', 'label' or 'both'.

## 4. Use cases

### 3.1 Tagging a single EEGLAB EEG structure

The cTagger stores the tag information, including the HED hierarchy, as a JSON string in:

EEG.etc.eventHedTags

To tag a single EEG structure, you can use the tagEEG static method of cTagger:

EEG = cTagger.tagEEG(EEG, eTagsBase, useGUI);

Initially, the EEG structure will not be tagged and so its field EEG.etc.eventHedTags will not exist. The tagEEG method extracts all of the unique event types from EEG.event and EEG.urevent and creates an eventTags object representing this EEG recording. The object includes the default Hed XML hierarchy when creating the eventTags object.

The eTagsBase argument is an eventTags object representing the results of tagging of other datasets that the user wants to propagate to this EEG structure. The value of the Hed XML hierarchy is merged with that of object representing the EEG structure. Similarly, the events of eTagsBase are merged using the matching strategy specified in eTagsBase as follows. If an event in eTagsBase also appears in the structure, the eTagsBase tags for that event are propagated. If the event does not appear at all in the EEG, the event is not propagated.

**Example:** Tag an EEG structure that has not previously been tagged using the GUI.

EEG = cTagger.tagEEG(EEG, '', true);

This function extracts the event information from the EEG structure and uses it, along with the default Hed XML to populate the GUI. If the user clicks submit on the GUI, the results will be stored in EEG.etc.eventHedTags. If the user clicks cancel, the EEG will not be changed.

### 3.2 Propagating tagging from one EEG structure to another

The second use case involves propagating the tagging information from one EEG structure to another without using the GUI. This is useful for a group of items that are similarly tagged.

**Example:** EEG1 is a structure that has already been tagged. The following code uses it to tag EEG2 without bringing up the GUI at all.

eTags1 = ctagger.getEEGEventTags(EEG1);

EEG2 = cTagger.tagEEG(EEG2, eTags1, false);

After this code has been executed, the EEG2 structure will have a EEG.etc.eventHedTags field that contains all of the event types of EEG2 with the tagging from EEG1 for the event types that match. Any updates to the Hed XML structure performed for EEG1 will also be propagated.

### 3.3 Simultaneously tagging all the EEG .set files in a directory tree

Often a researcher will collect a series of datasets using a variation of an experimental paradigm and the name of events and the tagging of those events should be maintained consistently across the datasets. It is possible to

### 3.4 Simultaneously tagging all the EEG datasets in a study

The base cTagger GUI is currently available and undergoing user testing. The cTagDB is currently being developed and tested.

### 3.5 Using cTagger with EEGLAB

The base cTagger GUI is currently available and undergoing user testing. The cTagDB is currently being developed and tested.

## 4. Status and availability

The base cTagger GUI is currently available and undergoing user testing. The cTagDB is currently being developed and tested.

## References

N. Bigdely-Shamlo, M. Miyakoshi, M. Westerfield, T. Bel-Bahar, S. Makeig (2012). Hierarchal event descriptor (HED) tags for meta-analysis of event-related EEG studies.