HED specification

Release 8.0.0

HED Working Group

Nov 29, 2021

CONTENTS:

A PDF version of this document can be found here

CHAPTER

ONE

1. INTRODUCTION TO HED

This document contains the specification for third generation HED or HED-3G. It is meant for the implementers and users of HED tools. Other tutorials and tagging guides are available to researchers using HED to annotate their data. This document contains the specification for the first official release of HED-3G (HED versions 8.0.0-xxx and above.) **When the term HED is used in this document, it refers to third generation (HED-3G) unless explicitly stated otherwise.**

The aspects of HED that are described in this document are supported or will soon be supported by validators and other tools and are available for immediate use by annotators. The schema vocabulary can be viewed using an expandable schema viewer.

All HED-related source and documentation repositories are housed on the HED-standard organization GitHub site, https://github.com/hed-standard, which is maintained by the HED Working Group. HED development is open-source and community-based. Also see the official HED website https://www.hedtags.org for a list of additional resources.

The HED Working Group invites those interested in HED to contribute to the development process. Users are encouraged to use the *Issues* mechanism of the `hed-specification` repository on the GitHub `hed-standard` working group website: https://github.com/hed-standard/hed-specification/issues to ask for help or make suggestions. The HED discussion forum https://github.com/hed-standard/hed-specification/discussions is maintained for in depth discussions of HED issues and evolution.

Several other aspects of HED annotation are being planned, but their specification has not been fully determined. These aspects are not contained in this specification document, but rather are contained in ancillary working documents which are open for discussion. These ancillary specifications include the HED working document on spatial annotation and the HED working document on task annotation.

1.1 1.1. Scope of HED

HED (an acronym for Hierarchical Event Descriptors) is an evolving framework that facilitates the description and formal annotation of events identified in time series data, together with tools for validation and for using HED annotations in data search, extraction, and analysis. HED allows researchers to annotate what happened during an experiment, including experimental stimuli and other sensory events, participant responses and actions, experimental design, the role of events in the task, and the temporal structure of the experiment. The resulting annotation is machine-actionable, meaning that it can be used as input to algorithms without manual intervention. HED facilitates detailed comparisons of data across studies.

As the name HED implies, much of the HED framework focuses on associating metadata with the experimental timeline to make datasets analysis-ready and machine-actionable. However, HED annotations and framework can be used to incorporate other types of metadata into analysis by providing a common API (Application Programming Interface) for building inter-operable tools.

This specification describes the official release of third generation of HED or HED-3G, which is HED version 8.0.0. Third generation HED represents a significant advance in documenting the content and intent of experiments in a format

that enables large-scale cross-study analysis of time-series behavioral and neuroimaging data, including but not limited to EEG, MEG, iEEG, fMRI, eye-tracking, motion-capture, EKG, and audiovisual recording.

HED annotations may be included in BIDS (Brain Imaging Data Structure) datasets https://bids.neuroimaging.io as described in *Chapter 6: Infrastructure*.

1.2 1.2. Brief history of HED

HED was originally proposed by Nima Bigdely-Shamlo in 2010 to support annotation in HeadIT an early public repository for EEG data hosted by the Swartz Center for Computational Neuroscience, UCSD (Bigdely-Shamlo et al. 2013). HED-1G was partially based on CogPO (Turner and Laird 2012).

Event annotation in HED-1G was organized around a single hierarchy whose root was the *Time-Locked Event*. Users could extend the HED-1G hierarchy at its deepest (leaf) nodes. First generation HED (HED-1G, versions < 5.0.0) attempted to describe events using a strictly hierarchical vocabulary.

HED-1G was oriented toward annotating stimuli and responses, but its lack of orthogonality in vocabulary design presented major difficulties. If *Red/Triangle* and *Green/Triangle* are terms in a hierarchy, one is also likely to need *Red/Square* and Green/Square* as well as other color and shape combinations.

HED-2G (versions 5.0.0 - 7.x.x) introduced a more orthogonal vocabulary, meaning that independent terms were in different subtrees of the vocabulary tree. Separating independent concepts such as shapes and colors into separate hierarchies, eliminates an exponential vocabulary growth due to term duplication in different branches of the hierarchy.

Parentheses were introduced so that terms could be grouped. Tools for validation and epoching based on HED tags were built, and large-scale cross-study "mega-analyses" were performed. However, as more complicated and varied datasets were annotated using HED-2G, the vocabulary started to become less manageable as HED tried to adapt to more complex annotation demands.

In 2019, work began on a rethinking of the HED vocabulary design, resulting in the release of the third generation of HED (HED-3G) in August 2021. HED-3G represents a dramatic increase in annotation capacity, but also a significant simplification of the user experience.

New in HED (versions 8.0.0+).

1. Improved vocabulary structure
2. Short-form annotation
3. Library schema
4. Definitions
5. Temporal scope
6. Encoding of experimental design

Following basic design principles, the HED Working Group redesigned the HED vocabulary tree to be organized in a balanced hierarchy with a limited number of subcategories at each node. (See the expandable schema browser to browser the vocabulary and explore the overall organization. *Chapter2:Terminology* defines some important HED tags and terminology used in HED.)

A major improvement in vocabulary design was the adoption of the requirement that individual nodes or terms in the HED vocabulary must be unique. This allows users to use individual node names (short form) rather than the full paths to the schema root during annotation, resulting in substantially simpler, more readable annotations.

To enable and regulate the extension process, the root HED-3G head schema specified here includes, for the first time, *HED library schema* to extend the HED vocabulary to include terms and concepts of importance to individual user

communities – for example researchers who design and perform experiments to study brain and language, brain and music, or brain dynamics in natural or virtual reality environments. The HED library schema concept may also be used to extend HED annotation to encompass specialized vocabularies used in clinical research and practice.

HED-3G also introduced a number of advanced tagging concepts that allow users to represent events with temporal duration, as well as annotations that represent experimental design.

1.3 1.2. Goals of HED

An event is a process that unfolds over time representing something that happens. Events are typically measured by noting sequences of time points (event markers) usually marking specific transition points which could be thought of as moments of phase transition in a dynamic process. HED annotation documents what happens at these event markers in order to facilitate data analysis and interpretation. Commonly recorded event markers in electrophysiological data collection include the initiation, termination, or other features of **sensory presentations** and **participant actions**. Other events may be **unplanned environmental events** (for example, noise and vibration from construction work unrelated to the experiment, or a laboratory device malfunction), **changes in experiment control** parameters as well as **data features** and control **mishaps** that cause operation to fall outside of normal experiment parameters. The goals of HED are to provide a standardized annotation and supporting infrastructure.

Goals of HED.

1. **Document the exact nature of events** (sensory, behavioral, environmental, and other) that occur during recorded time series data in order to inform data analysis and interpretation.
2. **Describe the design of the experiment** including participant task(s).
3. **Relate event occurrences** both to the experiment design and to participant tasks and experience.
4. **Provide basic infrastructure** for building and using machine-actionable tools to systematically analyze data associated with recorded events in and across data sets, studies, paradigms, and modalities.

A central goal of HED is to enable building of archives of brain imaging data in a form amenable to new forms of larger scale analysis, both within and across studies. Such event-related analysis requires that the nature(s) of the recorded events be specified in a common language. The HED project seeks to formalize the development of this language, to develop and distribute tools that maximize its ease of use, and to inform new and existing researchers of its purpose and value.

Most experiments have a limited number of distinct event types, which are often identified in the original experiment by local event codes. The strategy for assigning local codes to individual events depends on the format of the data set. However, in practice, HED tagging usually involves annotating a few event types or codes for an entire study, not tagging individual instances of events in individual data recordings.

1.4 1.3. HED design principles

The near decade-long effort to develop effective event annotation for neurophysiological and behavioral data, culminating to date in HED-3G, has revealed the importance of four principles (aka the PASS principles), all of which have roots in other fields:

The PASS principles for HED design.

1. **Preserve orthogonality** of concepts in specifying vocabularies.
2. **Abstract functionality** into layers (e.g., more general vs. more specific).

3. **Separate content** from presentation.
4. **Separate implementation** from the interface (for flexibility).

Orthogonality, the notion of keeping independently applicable concepts in separate hierarchies (1 above), has long been recognized as a fundamental principle in reusable software design, distilled in the design rule: *Favor composition over inheritance* (Gamma et al. 1994).

Abstraction of functionality into layers (2) and separation of content from presentation (3) are well-known principles in user-interface and graphics design that allow tools to maintain a single internal representation of needed information while emphasizing different aspects of the information when presenting it to users.

Similarly, making validation and analysis code independent of the HED schema (4) allows redesign of the schema without having to re-implement the annotation tools. A well-specified and stable API (application program interface) empowers tool developers.

1.5 1.4. Specification organization

This specification is meant to provide guidelines for tool-builders as well as HED annotators. *Chapter 2: Terminology* reviews the basic terminology used in HED, and *Chapter 3: Schema* outlines the rules for HED vocabularies. Basic and advanced event models and their annotations are explained in *Chapter 4: Basic annotation* and *Chapter 5: Advanced annotation*. Discussions of how tags for local event codes are associated with event instances are deferred to *Chapter 6: Infrastructure*.

HED provides a mechanism for user communities to develop discipline-specific library vocabularies. (See *Chapter 7: Library schema* for details.)

Appendix A: Schema format provides a reference manual for the HED vocabulary format rules. *Appendix B: HED errors* gives a complete listing of HED error codes and their meanings.

Other resources include a comprehensive list of HED *Documentation* resources and a list of *HED tools and services*.

All HED source code and resources are open-source and staged in the HED Standards Organization Repository https://github.com/hed-standard.

CHAPTER

TWO

2. HED TERMINOLOGY

The keywords “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in [RFC2119].

This specification uses a list of terms and abbreviations whose meaning is clarified here. Note: We here hyphenate multi-word terms as they appear in HED strings themselves; in plain text usage they may not need to be hyphenated. Starred variables [*] correspond to actual HED tags.

2.1 Agent [*]

A person or thing, living or virtual, that produces (or appears to participants to be ready and capable of producing) specified effects. Agents include the study participants from whom data is collected. Virtual agents may be human or other actors in virtual-reality or augmented-reality paradigms or on-screen in video or cartoon presentations (e.g., an actor interacting with the recorded participant in a social neuroscience experiment, or a dog or robot active in a live action or animated video).

2.2 Condition-variable [*]

An aspect of the experiment that is set or manipulated during the experiment to observe an effect or to manage bias. Condition variables are sometimes called independent variables.

2.3 Control-variable [*]

An aspect of the experiment that is fixed throughout the study and usually is explicitly controlled.

2.4 Dataset

A set of neuroimaging and behavioral data acquired for a purpose of a particular study. A dataset consists of data recordings acquired from one or more subjects, possibly from multiple sessions and sensor modalities. A dataset is often referred to as a study.

2.5 Event [*]

Something that happens during the recording or that may be perceived by a participant as happening, to which a time of occurrence (most typically onset or offset) can be identified. Something expected by a participant to happen at a certain time that does not happen can also be a meaningful recording event. The nature of other events may be known only to the experimenter or to the experiment control application (e.g., undisclosed condition changes in task parameters).

2.6 Event-context [*]

Circumstances forming or contributing to the setting in which an event occurs that are relevant to its interpretation, assessment, and consequences.

2.7 Event marker

A time point relative to the experimental timeline that can be associated with an annotation. Often such a marker indicates a transition point for some underlying event process.

2.8 Event-stream [*]

A named sequence of events such as all the events that are face stimuli or all of the events that are participant responses.

2.9 Experiment-participant [*]

A living agent, particularly a human from whom data is acquired during an experiment, though in some paradigms other human participants may also play roles.

2.10 Experimental-trial [*]

A contiguous data period that is considered a unit used to observe or measure something, typically a data period including an expected event sequence that is repeated many times during the experiment (possibly with variations). Example: a repeating sequence of stimulus presentation, participant response action, and sensory feedback delivery events in a sensory judgment task.

2.11 HED schema [*]

A formal specification of the vocabulary and rules of a particular version of HED for use in annotation, validation, and analysis. A HED schema is given in XML (`.xml`) format. The top-level versioned HED schema is used for all HED event annotations. Named and versioned HED library schema may be used as well to make use of descriptive terms used by a particular research community. (For example, an experiment on comprehension of connected speech might annotate events using a grammatical vocabulary contained in a linguistics HED schema library.)

2.12 HED string

A comma-separated list of HED tags and/or tag-groups.

2.13 HED tag

A valid path along one branch of a HED vocabulary hierarchy. A valid long-form HED tag is a slash-separated path following the schema tree hierarchy from its root to a term along some branch. Any suffix of a valid long-form HED tag is a valid short-form HED tag. No white space is allowed within terms themselves. For example, the long form of the HED tag specifying an experiment participant is: *Property/Agent-property/Agent-task-role/Experiment-participant*. Valid short-form tags are *Experiment-participant*, *Agent-task-role/Experiment-participant*, and *Agent-property/Agent-task-role/Experiment-participant*. HED tools should treat long-form and short-form tags interchangeably.

2.14 Indicator-variable [*]

An aspect of the experiment or task that is measured or calculated for analysis. Indicator variables, sometimes called dependent variables, can be data features that are calculated from measurements rather than aspects that are directly measured.

2.15 Parameter [*]

An experiment-specific item, often a specific behavioral or computer measure, that is useful in documenting the analysis or assisting downstream analysis.

2.16 Recording [*]

A continuous recording of data from an instrument in a single session without repositioning the recording sensors.

2.17 Tag-group

One or more valid, comma-separated HED tags or enclosed in parentheses to indicate that these tags belong together. Tag-groups may contain arbitrary nestings of other tags and tag-groups.

2.18 Task [*]

A set of structured activities performed by the participant that are integrally related to the purpose of the experiment. Tasks often include observations and responses to sensory presentations as well as specified actions in response to presented situations.

2.19 Temporal scope

The time interval between events marking the beginning and end of something in the experiment. The time between and including the onset and offset of an event.

2.20 Time-block [*]

A contiguous portion of the data recording during which some aspect of the experiment is fixed or noted.

CHAPTER

THREE

3. THE HED SCHEMA

A HED schema is the formal specification of the HED vocabulary and rules for annotating events. The HED schema vocabulary is organized hierarchically so that similar concepts and terms appear close to one another in the organizational hierarchy. (See the expandable schema viewer.) Valid HED annotations must be drawn from a HED schema vocabulary and HED validators and other tools use the information encoded in the relevant schema when performing validation and other processing of HED annotations.

Users must provide the version of the HED schema they are using when creating an annotation. Past, present, and future versions of the HED schema adhere to semantic versioning with version numbers of the form *x.y.z* representing *major.minor.patch* versions. Although schema developers work with HED schema in `.mediawiki` format for ease in editing, HED tools generally use XML versions of the HED schema.

Standard development process for XML schema.

1. Create or modify a `.mediawiki` file containing the schema.
2. Convert to `.xml` using the HED tools.
3. View in the expandable schema viewer to verify.

HED schema XML filenames use the standardized format `HEDx.y.z.xml`. These standardized names make it easier for tools to locate the appropriate HED schema version in the HED working group GitHub website. The XML schema versions are stored in the `hedxml` directory of the HED specification repository.

Third generation HED begins with schema version 8.0.0. Thus, the file containing the first official release of the third generation HED schema is `HED8.0.0.xml`. Note: HED versions *8.0.0-alpha.1* through *8.0.0-beta.5* are prerelease versions of HED-3G and should not be used as they will eventually be deprecated.

Releases are stored in `hedxml` directory of the `hed-specification` repository. Deprecated versions of the HED schema are stored in the `hedxml/deprecated` directory of the `hed-specification` repository.

All data recordings in a dataset should be annotated using a single version of the standard HED schema. Validation and analysis tools are not expected to handle multiple versions of the standard HED schema when processing a dataset. Datasets may also include annotations from multiple HED library schema extensions in addition to those from the standard schema, as described in *Chapter 7: Library schema* of this document. A more detailed discussion of the HED schema format appears in *Appendix A*.

3.1 3.1. Mediawiki schema format

HED schema developers usually specify schema in `.mediawiki` format for more convenient editing, display, and reference on GitHub. However, tools assume that the schema is in .mediawiki format. Conversion tools allow The following brief example illustrates the format. A full description of the format is given in *Appendix A*.

Example: Layout of a HED schema (.mediawiki).

```
HED version="8.0.0"

'''Prologue'''
This prologue introduces the schema.

!# start schema
'''Event''' <nowiki>[Something that happens at a given place and time.]</nowiki>
* Sensory-event <nowiki>{suggestedTask=Task-event-role}[Something perceivable by an␣
↪agent.]</nowiki>
                                . . .
'''Property'''<nowiki>{extensionAllowed}[A characteristic.] </nowiki>
* Informational-property <nowiki>[A quality pertaining to information.]</nowiki>
** Label <nowiki>{requireChild} [A string of 20 or fewer characters.]</nowiki>
*** <nowiki># {takesValue, valueClass=nameClass}</nowiki>
!# end schema

'''Unit classes''' <nowiki>[Unit classes and units for the nodes.]</nowiki>
                            . . .
'''Unit modifiers''' <nowiki>[Unit multiples and submultiples.]</nowiki>
                            . . .
'''Value classes''' <nowiki>[Rules for the values provided by users.]</nowiki>
                            . . .
'''Schema attributes''' <nowiki>[Allowed node attributes.]</nowiki>
                            . . .
'''Properties''' <nowiki>[Properties of the schema attributes.]</nowiki>
                            . . .
'''Epilogue'''
An optional section that is the place for notes and is ignored in HED processing.

!# end hed
```

Beginning with third generation HED (HED schema versions 8.0.0 and later), **terms in a given schema must be unique within that schema.** This uniqueness rule allows automated expansion of short form HED strings into their full long forms.

Top level tree root elements are enclosed by triple single quotes. Each child term within the schema must be on a single line that begins with a certain number of consecutive asterisks (*) corresponding to the term's level within the hierarchy. The term or node name is separate from its level-indicating asterisks by a space.

Everything after each HED term must be enclosed by <nowiki></nowiki> markup elements. Items within these markup elements include a term description and term attributes.

Term (node element) descriptions are enclosed in square brackets ([ ]) in the `.mediawiki` specification and indicate the meaning of the term or tag they modify.

HED term attributes are enclosed with curly braces ({ }). These term attribute provide additional rules about how the

tag and modifying values should be used and handled by tools. Allowed HED term attributes include unit class and value class values as well as HED schema attributes that do not have the `unitClassProperty`, `unitModifierProperty`, `unitProperty`, or `valueClassProperty`.

HED term attributes appear in the schema specification either as `name` attributes or as `name=value` pairs. The presence of a `name` attribute for a schema node element indicates that the attribute is true for that term, while the presence of a `name=value` attribute indicates that the attribute has the specified value for that term. If multiple values of a particular attribute are applicable, they should be specified as separate name-value pairs.

The hashtag character (#) is a placeholder for a user-supplied value. Within the HED schema a # node indicates that the user must supply a value consistent with the unit classes and value classes of the # node if it has any. Lines with hashtag (#) placeholders should have everything after the asterisks enclosed by <nowiki></nowiki> markup elements. The values of HED tag placeholders cannot stand alone, but must include the parent when used in a HED string.In the above example, the # that is a child of the *Label* node must include *Label* when used (e.g., *Label/myLabel*).

3.2 3.2. XML schema format

The HED XML version of the schema is used during validation and analysis. The `.xml` format has changed with the release of HED-3G. This modification of the XML format was done for the following reasons.

Reasons for XML file format change for HED.

1. To correctly handle multiple values of schema attributes.
2. To preserve the prologue and epilogue information present in `.mediawiki` files.
3. To allow schema attributes to be formally specified and validated.
4. To allow an XSD specification of the HED schema for validation of the schema.

The following is a translation of the `.mediawiki` example from the previous section in the new XML format. A complete specification of the format is given in *Appendix A: Schema format*.

Example: XML version of previous example.

```
<?xml version="1.0" ?>
<HED version="8.0.0">
    <prologue>This prologue introduces the schema.</prologue>
    <schema>
        <node>
           <name>Event</name>
           <description>Something that happens at a given place and time.</description>
           <node>
               <name>Sensory-event</name>
               <description>Something perceivable by an agent.</description>
               <attribute>
                   <name>suggestedTag</name>
                   <value>Task-event-role</value>
               </attribute>
           </node>
        </node>
                        . . .
        <node>
```

(continues on next page)

(continued from previous page)

```
            <name>Property</name>
            <description>A characteristic of some entity.</description>
            <attribute>
                <name>extensionAllowed</name>
            </attribute>
            <node>
                <name>Informational-property</name>
                <description>A quality pertaining to information.</description>
                <node>
                    <name>Label</name>
                    <description>A string of less than 20.</description>
                    <attribute>
                        <name>requireChild</name>
                    </attribute>
                <node>
                    <name>#</name>
                    <attribute>
                        <name>takesValue</name>
                    </attribute>
                    <attribute>
                        <name>valueClass</name>
                        <value>nameClass</value>
                    </attribute>
                </node></node>
              </node>
        </node>
    </schema>
    <unitClassDefinitions> ...</unitClassDefinitions>
    <unitModifierDefinitions>...</unitModifierDefinitions>
    <valueClassDefinitions>...</valueClassDefinitions>
    <schemaAttributeDefinitions>...</schemaAttributeDefinitions>
    <propertyDefinitions>...</propertyDefinitions>
    <epilogue>This epilogue is a place for notes and is ignored in HED processing.</
↪epilogue>
</HED>
```

3.2.1 3.2.1 The <node> element

Each <node> element must have a <name> child element corresponding to the HED tag term that it specifies. A <node> element may also have a <description> child element containing the text that appears in square brackets ([ ]) in the .mediawiki version. The schema attributes (which appear as name values or name-value pairs enclosed in curly braces {} in the .mediawiki file) are translated into <attribute> child elements of <node> in the .xml.

3.2.2 3.2.2 Unit classes and modifiers

The HED schema also includes a <unitClassDefinitions> section that specifies the allowed unit classes and the corresponding allowed unit names. Only the singular version of each unit name is explicitly specified, but the corresponding plurals of the explicitly mentioned singular versions are also allowed (e.g., feet is allowed in addition to foot). HED uses a pluralize function available in both Python and Javascript to check validity.

The <unitModifierDefinitions> section lists the SI unit multiples and submultiples that are allowed to be prepended to units that have the SIUnit schema attribute.

3.2.3 3.2.3 Value classes

The <valueClassDefinitions> section specifies rules for the values that are substituted for placeholders (#). Examples are special characters that are allowed for numeric values or dates. Placeholders that have no valueClass attributes, are assumed to follow the rules for HED tag naming described in the next section.

3.2.4 3.2.4 Schema attributes

The <schemaAttributeDefinitions> section lists the schema attributes that apply to some nodes and definitions in other sections of the schema. The specification of which type of elements an attribute may apply to is specified by the property attributes of these schema attributes.

3.2.5 3.2.5 Schema properties

The <schemaPropertyDefinitions> section lists properties of the schema attributes, themselves. This specification allows general validation to handle a lot of the processing directly based on the HED schema contents rather than on hard-coded implementation.

3.3 3.3. Allowed characters

The different parts of the HED schema and associated HED tags have different rules for the characters that are allowed. UTF-8 characters are not supported. Schema designers and users that extend HED schema must use node or term names that conform to the rules for valueClass=nameClass. Placeholder values that don't have an associated valueClass attribute are also assumed to have valueClass=nameClass.

Table 1: Rules for valid HED characters.

Element	Allowed characters
Node (nameClass)	Alphanumeric characters, hyphens, and underbars with no white space.
Description (textClass)	Alphanumeric characters, blanks, commas, periods, semicolons, colons, hyphens, underbars, forward slashes, carets (^), and parentheses.
Placeholder (#)	A special node value which indicates a later substitution.
Placeholder children	Depends on valueClass as well as allowed unitClass and unit modifiers.
Library names	A single word containing only alphabetic characters.
Namespaces	A single alphabetic word followed by a single colon.

Notes on rules for allowed characters in the HED schema.

1. The first letter of a term should be capitalized with the remainder lower case.

2. Terms containing multiple words cannot contain blanks and should be hyphenated.
3. Blanks around comma and parentheses delimiters are not part of a tag.
4. Descriptions should be concise sentences, possibly with clarifying examples.
5. Descriptions cannot contain square brackets, curly braces, quotes, or punctuation not specifically allowed by `textClass`.
6. Values substituted for # may have special characters determined by the value class. For example, the colon (:) is specifically allowed for the `dateTimeClass` value class.
7. Units are separated from their value by at least one blank whether prefix or suffix.
8. Library namespace names are local and consist of a short alphabetic word followed by a single colon.

3.3.1 3.3.2 Placeholder values

Blanks are allowed as are periods, dollar ($), percent (%), caret (^), plus (+), minus(-), under bar(_), and semicolon (;). Values must conform to the underlying unit classes of the placeholder specification.

Certain unit classes allow other special characters in their value specification. These special characters are specified in the schema with the `allowedCharacter` attribute. An example of this is the colon in the `dateTimeClass` unit class.

3.4 3.4. Vocabulary organization

The HED-3G schema (version 8.0.0 and above) contains six root trees of HED terms: *Event*, *Agent*, *Action*, *Item*, *Property*, and *Relation*.

The *Event* root tree terms indicate the general category of the event, such as whether it is a sensory event, an agent action, a data feature, or an event indicating experiment control or structure. The HED annotations describing each event may be assembled from a number of sources during processing.

The assembled HED string annotating an event should have at least one tag from the *Event* tree, as many analysis tools use the *Event* tags as a primary means of segregating, epoching, and processing the data. Ideally, tags from the *Event* subtree should appear at the top level of the HED annotation describing an event to facilitate analysis.

The *Agent* root tree terms indicate types of agents (e.g., persons, animals, avatars) that take an active role or produce a specified effect. An *Agent* tag should be grouped with property tags that provide information about the agent, such as whether the agent is an experiment participant.

The *Action* root tree terms describe actions performed by agents. Generally these are grouped in a triple *(A, (Action, B))* which is interpreted as *A* does *Action* on *B*. If the action does not have a target, it should be annotated *(A, (Action))*, meaning *A* does *Action*.

The *Item* root tree terms describe things with (actual or virtual) physical existence such as objects, sounds, or language.

Descriptive elements are organized in the *Property* rooted tree. These descriptive elements should always be grouped with the elements they describe using parentheses.

Binary relations are in the *Relation* rooted tree. Like items from the *Action* sub-tree, these should be annotated using *(A, (Relation, B))*.

3.5 3.5. Tag syntax

A **HED tag** is a term in the HED vocabulary identified by a path consisting of the individual node names from some branch of the HED schema hierarchy separated by forward slashes (/). An important requirement of third generation HED is that the node names in the HED schema **must be unique**. As a consequence, the user can specify as much of the path to the root as desired. The full path version is referred to as **long form** and truncated versions as **short form**. HED tools are available to map between shortened tags and long form as needed. Any intermediate form of the tag path is also allowed as illustrated by this example:

Example: Equivalent forms for HED tag representing a triangle.

1. *Item/Object/Geometric-object/2D-shape/Triangle*
2. *Object/Geometric-object/2D-shape/Triangle*
3. *Geometric-object/2D-shape/Triangle*
4. *2D-shape/Triangle*
5. *Triangle*

For values that are substituted for a placeholder (#) child, the tag must include the parent as illustrated in this example for the *Label* tag. The values that replace these # placeholders cannot be node names.

Example: Equivalent forms for HED tag representing the label Image1.

1. *Property/Informational-property/Label/Image1*
2. *Informational-property/Label/Image1*
3. *Label/Image1*

A **HED string** is a comma-separated list of HED tags and/or HED tag groups. A **HED tag group** is a comma-separated list of HED tags and/or tag groups enclosed in parentheses. Tag groups may include other tag groups. Parentheses convey association, since HED strings are unordered lists. The terms in a HED string must be unique, thus, a HED string forms a set.

Example: Nested HED tag group indicated press.

Short form:

((Human-agent, Experiment-participant), (Press, Mouse-button))

Long form:

((Agent/Human-agent,Property/Agent-property/Agent-task-role/Experiment-participant),(Action/Move/Move-body-part/Move-upper-extremity/Press,Item/Object/Man-made-object/Device/IO-device/Input-device/Computer-mouse/Mouse-button))

The validation errors for HED tags and HED strings are summarized in *Appendix B: HED errors*.

HED # placeholders cannot have siblings. Thus, tags that have placeholder children cannot be extended even if they inherit an `extensionAllowed` attribute from an ancestor. The parsers treat any child of these tags as a value rather than a tag.

HED values can be strings or numeric values followed by a unit specification. If a `unitClass` is specified as an attribute of the `#` node, then the units specified must be valid units for that `unitClass`. **HED parsers assume that units are separated from values by at least one blank.**

CHAPTER

FOUR

4. BASIC ANNOTATION

This section illustrates the use of HED tags and discusses various tags that are used to document the structure and organization of electrophysiological experiments. The simplest annotations treat each event as happening at a single point in time. The annotation process for such events involves describing what happened during that event.

This chapter illustrates basic HED descriptions of four types of events that are often annotated using single event markers: **stimulus events**, **response events**, **experiment control events**, and **data features**.

HED-3G now also allows more sophisticated models of events that unfold over time using multiple event markers. Downstream analyses often look for neurological effects directly following (or preceding) event markers. The addition of HED context, allows information about events that occur over extended periods of time to propagate to intermediate time points. *Chapter 5: Advanced annotation* develops the HED concepts needed to capture these advanced models of events as well as event and task inter-relationships.

4.1 4.1. Instantaneous events

This section describes HED annotation of events that are modeled as happening at an instant in time. Sometimes the event marker corresponding to such an event is inserted in the data or held in an external event file containing the onset time of some action, relative to the beginning of the data recording. We refer to these events as **time-marked events**. The event marker may also point to the end/offset of some happening or to time between the onset and offset (for example, the maximum velocity point in a participant arm movement or the maximum potential peak of an eye-blink artifact).

A typical example of an experiment using time-marked event annotation is simple target detection. In this experiment geometric shapes of different colors are presented on a computer screen at two-second intervals. After every visual shape presentation, the subject is asked to press the left mouse button if the shape is a green triangle or the right mouse button otherwise. After a block of 30 such presentation-response sequences (trials), the control software sounds a buzzer to indicate that the subject can rest for 5 minutes before continuing to the next block of trials. After the experiment is completed, the experiment runs an eyeblink-detection tool on the EEG data and inserts an event marker at the amplitude maximum of each detected blink artifact.

4.2 4.2. Sensory presentations

The target detection experiment described above is an example of a stimulus-response paradigm: perceptually distinct sensory stimuli are presented at precisely recorded times (typically with abrupt onsets) and ensuing and/or preceding precisely-timed changes in the behavioral and physiological data streams are annotated or analyzed. Stimulus onsets (typically) are annotated with the *Sensory-event* tag. Additional tags indicate task role. Separation of what an event is (as designated by a tag from the *Event* subtree) from its task role (as indicated by other descriptive tags) is an important design change that distinguishes HED-3G from earlier versions of HED and enables effective annotation in more complex situations.

A stimulus event can be annotated at different levels of detail. When not needed, fine details can generally be ignored, but once annotated can provide valuable information for later, possibly unanticipated analysis purposes. In a series of examples, we will annotate successively more details about the experiment events. Each example shows both the short form and long form. The elements in the long form that correspond to the short form are shown in bold-face. In addition, the long form has a description, which is omitted from the short-form for readability.

The following example illustrates a very basic annotation of a stimulus event, indicating the stimulus is a green triangle presented visually. The annotation indicates that this is a visual sensory event intended to be an experiment stimulus. *Sensory-event* is in the *Event* rooted tree and indicates the general class that this event falls into.

Example: Version 1 of a visual stimulus annotation.

Short form:

Sensory-event, Experimental-stimulus, Visual-presentation, (Green, Triangle)

Long form:

Event/Sensory-event,Property/task-property/Task-event-role/Experimental-stimulus,Property/Sensory-property/Visual-presentation,(Property/Sensory-property/Sensory-attribute/Visual-attribute/Color/CSS-color/Green-color/Green,Item/Object/Geometric-object/2D-shape/Triangle),Property/Informational-property/Description/An experimental stimulus consisting of a green triangle is displayed on the center of the screen.

The example HED string above illustrates the most basic form of point event annotation. In general, the annotation for each event should include at least one tag from the *Event* tree. If there are multiple sensory presentations in the same event, a single *Sensory-event* tag covers the general category for all presentations in the event. The individual presentations (which may include different modalities) are grouped with their descriptive tags, while the *Sensory-event* tag applies overall. In this case there is only one, so the grouping is not necessary.

The *Experimental-stimulus* is a *Task-property* tag. Whether a particular sensory event is an experiment stimulus depends on the particular task, hence *Experimental-stimulus* is a *Task-property*. Sensory events that are extraneous to the task can also occur, so it is important to distinguish those that are related to the intent of the task.

The remaining portion of the annotation describes what the sensory presentation is. The *Green* and *Triangle* tags are grouped to indicate specifically that a green triangle is presented. *Visual-presentation* is a *Sensory-property* tag from the *Property* rooted tree. Which senses are impacted by the *Sensory-event* should always be indicated, even if it appears to be obvious to the reader. The goal is to facilitate machine-actionable analysis.

HED has a number of qualitative relational tags designating spatial features such as *Center-of*, which should always be included if possible. These qualitative terms provide clear search anchors for tools looking for general positional characteristics. Hemispheric and vertical distinctions have particular neurological significance. More detailed size, shape, and position information enhances the annotation. However, actual detailed information requires the specification of a frame of reference, a topic deferred until later in this document.

The order of the tags does not matter. HED strings are unordered lists of HED tags and tag groups. Where the grouping of associated tags needs to be indicated, most commonly in the case of tags with modifiers, the related tags should be put in a tag group enclosed by parentheses (as above).

Notice that the long form version also includes a *Description* tag that gives a text description of the event. Users should always include a *Description* tag in the annotation of each event type. The *Description* tag is omitted for readability in the short form examples. As a matter of practice, however, users should start with a detailed text description of each type of event before starting the annotation. This description can serve as a check on the consistency and completeness of the annotation. Generally users annotate using the short form for HED tags and use tools to map the short form into the long form during validation or analysis.

4.3 4.3. Task role

In deciding what additional information should be included, the annotator should consider how to convey the nature and intent of the experiment and the EEG responses that are likely to be elicited. The brief description suggests that green triangles are something "looked for", within the structure of the task that participants are asked to perform during the experiment. The following annotation of the green triangle presentation includes information about the role this stimulus appears in the task.

Example: Version 2 of a visual stimulus annotation.

Short form:

Sensory-event, Experimental-stimulus, Visual-presentation,(Green, Triangle), (Intended-effect, Oddball), (Intended-effect, Target)

Long form:

Event/Sensory-event,Property/Task-property/Task-event-role/Experimental-stimulus,Property/Sensory-property/Sensory-presentation/Visual-presentation,(Property/Sensory-property/Sensory-attribute/Visual-attribute/Color/CSS-color/Green-color/Green,Item/Object/Geometric-object/2D-shape/Triangle),(Property/Task-property/Task-effect-evidence/Intended-effect,Property/Task-property/Task-stimulus-role/Oddball),(Property/Task-property/Task-effect-evidence/Intended-effect,Property/Task-property/Task-stimulus-role/Target),Property/Informational-property/Description/A green triangle target oddball is presentedin the center of the screen with probability 0.1.

The *Intended-effect* tag is a *Task-effect-evidence* tag that describes the effect expected to be elicited from the participant experiencing the stimulus. This tag indicates, that based on the specification of the task, we can conclude that the subject will be looking for the triangle (*Target*) and that its appearance is unusual (*Oddball*).

Three other tags in the *Task-effect-evidence* are *Computational-evidence*, *External-evidence*, and *Behavioral-evidence*. In many experiments, a subject indicates that something occurs by performing an action such as pushing the left mouse button for a green triangle and the right button otherwise. When the left-mouse button is pushed, one may conclude that the participant has behaved as though the green triangle appears. If the button push is tagged with *Behavioral-evidence*, automated tools can check whether the intended effect agrees with subject behavior. An example of *External-evidence* is annotation by a speech therapist about whether the participant stuttered in a speech experiment. *Computational-evidence* might be generated from BCI annotation.

HED-3G has more sophisticated methods of specifying the relationships of events and tasks. These require more advanced tagging mechanisms that are discussed later in this document.

4.4 4.4. Agent actions

In many experiments, the participant is asked to press (or select and press) a finger button to indicate their perception of or judgment concerning the stimulus. These types of events, as well as participant actions not related to the task, are annotated as *Agent-action* events. *Agent-action* events can be annotated with varying levels of detail, as illustrated by the next two examples.

Example: Version 1 of button press annotation.

Short form:

Agent-action, (Participant-response, (Press, Mouse-button))

The *Participant-response* tag indicates that this event represents a task-related response to a stimulus. The *Press* tag is from the *Action* subtree and is grouped with the *Mouse-button* to indicate the pressing of a button. In general, *Action* elements can be considered verbs, while *Items* and *Agents* can be considered nouns. These elements form a natural sentence structure: (subject, (verb, direct object)), with the subject and direct object being formed by noun elements. *Attribute* elements are the adjectives, adverbs, and prepositions that modify and connect these elements.

Example: Version 2 of a button press annotation.

Short form:

Agent-action, Participant-response,((Human-agent, Experiment-participant), (Press, Mouse-button)),(Behavioral-evidence, Oddball), (Behavioral-evidence, Target)

Long form:

Event/Agent-action,Property/Task-property/Task-event-role/Participant-response,((Agent/Human-agent,Property/Agent-property/Agent-task-role/Experiment-participant),(Action/Move/Move-body-part/Move-upper-extremity/Press,Item/Object/Man-made-object/Device/IO-Device/Input-device/Computer-mouse/Mouse-button)),(Property/Task-property/Task-effect-evidence/Behavioral-evidence,Property/Task-property/Task-stimulus-role/Oddball),(Property/Task-property/Task-effect-evidence/Behavioral-evidence,Property/Task-property/Task-stimulus-role/Target),Property/Informational-property/Description/The subject pushes the left mouse buttonto indicate the appearance of an oddball target using index finger on the left hand.

The *Participant-response* tag is modified by tags that indicate that the participant is reacting by responding as though the stimulus were an oddball target. Specifically the *Behavioral-evidence* tag documents that the subject gave a response indicating an oddball target. In other words, the participant pressed the left mouse button indicating an oddball target, which may or may not match the stimulus that was presented.

Other details should be annotated, including whether the subject's left, right, or dominant hand was used to press the mouse button and whether the left mouse button or right mouse button was pressed. (This factor was indicated in the *Description*, but not in the machine-actionable tags.)

4.5 4.5. Experimental control

Experiments may have experiment control events written into the event record, often automatically by the presentation or control software. In the illustration provided above, a buzzer sounded by the control software indicates that the subject should rest.

Example: Version 1 of a simple feedback event.

Short form:

Sensory-event, Instructional, Auditory-presentation,(Buzz, (Intended-effect, Rest))

Long form:

Event/Sensory-event,Property/Task-property/Task-event-role/Instructional,Property/Sensory-property/Sensory-presentation/Auditory-presentation,(Item/Sound/Named-object-sound/Buzz,(Property/Task-property/Task-effect-evidence/Intended-effect,Action/Perform/Rest)),Property/Informational-property/Description/A buzzer sounds indicating a rest period.

4.6 4.6. Data features

Another type of tagging documents computed data features and expert annotations that have been inserted post-hoc into the experimental record as events. The *Computed-feature* and *Observation* tags designate whether the event came from a computation or from manual evaluation. The following example illustrates a HED annotation from

---

Example: Annotation of an inserted computed feature.

Short form:

Data-feature, (Computed-feature, Label/Blinker_BlinkMax)

Long form:

Event/Data-feature,(Property/Data-property/Data-source-type/Computed-feature,Property/Informational-property/Label/Blinker_BlinkMax),Property/Informational-property/Description/Event marking the maximum signaldeviation caused by blink inserted by the Blinker tool.

As shown by this example, the *Computed-feature* tag is grouped with a label of the form *toolName_featureName*. In this example, the computed property is just a marker of where a feature was detected. If a value was computed at this point, an additional *Value* tag would be included.

Clinical evaluations are observational features and many fields have standardized names for these features. Although the HED standard itself does not specify these names, library schema representing terminology in clinical or application subfields may provide the vocabulary. (See *Chapter 7: Library schema* for a discussion of library schema.) The following example illustrates how annotation from a human expert can be annotated in HED.

Example: Annotator AJM identifies a K-complex in a sleep record.

Short form:

Data-feature, (Observation, Label/AnnotatorAJM_K-complex)

Long form:

Event/Data-feature,(Property/Data-property/Data-source-type/Observation,Property/Informational-property/Label/AnnotatorAJM_K-complex),Property/Informational-property/Description/K-complex defined by AASM guide.

4.7 4.7. What else?

Most event annotation focuses on basic identification and description of stimuli and the participant's direct response to that stimuli. However, for accurate comparisons across studies, much more information is required and should be documented with HED tags rather than just with text descriptions. This is particularly true if this information is relevant to the experimental intent, varied during the experiment, or likely to evoke a neural response.

The example of *Section 4.1:Instantaneous events*, the sensory presentation of a green triangle stimulus image models the event as happening at a single point in time. More realistically, the green triangle might be displayed for an extended period (during which other events might occur). Further, the disappearance of the triangle is likely to elicit a neural response. Exactly how this information should be represented is discussed in *Section 5.3: Temporal scope* with the introduction of temporal scope and its use with *Onset* and *Offset*.

Even for a standard setup, aspects such as the screen size, the distance and position of the participant relative to the screen and the stimulus, as well as other details of the environment, should be documented as part of the overall

experiment context. These details allow analysis tools to compare and contrast studies or to translate visual stimuli into visual field information. *Event-context* tags, which are introduced in *Section 5.5: Event context*, allow this information to be propagated to recording events in a manner that is convenient for analysis.

HED also allows the embedding of annotations for the design of the experiment, documenting how and when condition variables and other aspects of an experiment are changed.*Section 5.6: Experimental design* describes HED mechanisms for annotating this information.

CHAPTER

FIVE

5. ADVANCED ANNOTATION

5.1 5.1. HED definitions

HED-3G introduces the *Definition* tag to facilitate tag reuse and to allow implementation of concepts such as **temporal scope**. The *Definition* tag allows researchers to create a name to represent a group of tags and then use the name in place of these tags when annotating data. These short-cuts make tagging easier and reduce the chance of errors. Often laboratories have a standard setup and event codes with particular meanings. Researchers can define names and reuse them for multiple experiments. Another important role of definitions is to provide the structure for implementing temporal scope as introduced in *Section 5.3: Temporal Scope*.

A **HED definition** is a tag group that includes one *Definition* tag whose required child value names. The definition usually includes an optional tag-group specifying the actual definition information. The following summarizes the syntax of definition.

Syntax summary for *Definition*

Short forms: *(Definition/XXX, (tag-group))*

(Definition/XXX/#, (tag-group))

Long forms: *(Property/Organizational-property/Definition/XXX, (tag-group))*

(Property/Organizational-property/Definition/XXX/#, (tag-group))

Notes:

1. *XXX* is the name of the definition and *(tag-group)* is the definition's value.
2. If the *XXX/#* form is used, then the definition's *(tag-group)* MUST contain a single # representing a value to be substituted for when the definition is used.
3. The *tag-group* may be omitted if the only purpose of the definition is to define a label to anchor temporal scope. (*Chapter 5.3: Temporal Scope*).
4. The *tag-group* is required if the # placeholder is used.
5. Neither the definition name *XXX* nor the value substituted for the # placeholder can be node names.

The following example defines the *PlayMovie* term.

Example: *PlayMovie* represents playing a movie on the screen.

Short form:

(Definition/PlayMovie, (Visual-presentation, Movie, Computer-screen))

Long form:

(Property/Organization-property/Definition/PlayMovie,(Property/Sensory-property/Sensory-presentation/Visual-presentation,Item/Object/Man-made-object/Media/Visualization/Movie,Item/Object/Man-made-object/Device/IO-device/Output-device/Display-device/Computer-screen))

The placeholder form of the definition is used, for example, to annotate an experimental parameter whose value is selected at random for each occurrence. The annotator can use a single definition name and just substitute the value for each occurrence.

Example: Value definition to annotate the rate of visual presentation.

Short form:

(Definition/PresentationRate/#,(Visual-presentation, Experimental-stimulus, Temporal-rate/# Hz))

Long form:

(Property/Organizational-property/Definition/PresentationRate/#,(Property/Sensory-property/Sensory-presentation/Visual-presentation,Property/Task-property/Task-event-role/Experimental-stimulus,Data-property/Data-value/Spatiotemporal-value/Rate-of-change/Temporal-rate/#))

5.2 5.2. Using definitions

When a definition name such as *PlayMovie* or *PresentationRate* is used in an annotation, the name is prefixed by *Def/* to indicate that the name represents a defined name. In other words, *Def/PlayMovie* is shorthand for *(Visual, Movie, Screen)*. The following summarizes *Def/* syntax rules.

Syntax summary for *Def*

Short forms: *Def/XXX*

Def/XXX/#

Long forms: *Property/Organizational-property/Def/XXX*

Property/Organizational-property/Def/XXX/#

Notes:

1. *XXX* is the name of the definition.
2. If the *XXX/#* form is used, then the corresponding definition's *(tag-group)* MUST contain a single # representing a value to be substituted for when the definition is used.

The following example shows how a defined name is used in annotation.

Example: Use *PresentationRate* to annotate a presentation rate of 1.5 Hz.

Short form: *Def/PresentationRate/1.5 Hz*

Long form: *Property/Organizational-property/Def/PresentationRate/1.5 Hz*

During analysis, tools usually replace *Def/PlayMovie* with a fully expanded tag string. Tools must retain the association of the expanded tag string with the definition name for identification during searching and substitution.

When a definition is expanded, the resulting tag string should include the definition name using the *Def-expand* tag. In other words, the tools should expand the definition as *(Def-expand/PlayMovie, Visual, Movie, Screen)*. The *Def-expand/PlayMovie* is inserted in the definition tag group as part of the expansion to keep the association with the original definition.

Usually definitions do not contain tags from the *Event* subtree. The standard practice is to use the elements of the *Event* subtree as top-level tags to designate the general category of an event. This practice makes it easier for search and analysis tools to filter events without extensive parsing. The annotator can use tags such as *Experimental-stimulus* (Long form: *Property/Task-property/Task-event-role/Experimental-stimulus*) to explain the role of a particular sensory presentation element in the experiment within the definition.

Definitions may appear anywhere in a HED event file or in auxiliary files associating metadata with HED tags such as JSON sidecars in BIDS datasets. Multiple definitions can be defined or used in the same HED string annotation, but definitions cannot be nested. Further, definitions must appear as top-level tag groups. Tools generally make a pass through the event information to extract the definitions prior to other processing. The validation checks made by the HED validator when assembling and processing definitions are summarized in *Appendix B:*.

In addition to syntax checks, which occur in early processing passes, HED validators check that names are defined before they are used as definitions. Additional checks for temporal scope are discussed in *Section 5.3: Temporal scope*.

5.3 5.3. Temporal scope

Events are often modeled as instantaneous occurrences that occur at single points in time (i.e., time-marked or point events). In reality, many events unfold over extended time periods. The interval between the initiation of an event and its completion is called the **temporal scope** of the event. HED events are assumed to be point events unless they are given an explicit temporal scope (i.e., they are "scoped" events).

Some events, such as the setup and initiation of the environmental controls for an experiment, may have a temporal scope that spans the entire data recording. Other events, such as the playing of a movie clip or a participant performing an action in response to a sensory presentation, may last for seconds or minutes. Temporal scope captures the effects of these extended events in a machine-actionable manner. HED has two different mechanisms for expressing temporal scope: *Onset*/*Offset* and *Duration*.

5.3.1 5.3.1. *Onset* and *Offset*

The most direct HED method of specifies scoped events by combining *Onset* and *Offset* tags with defined names. Using this method, an event with temporal scope actually corresponds to two point events.

The initiation event is tagged by a *(Def/XXX, Onset)* where *XXX* is a defined name. The end of the event's temporal scope is marked either by a *(Def/XXX, Offset)* or by another *(Def/XXX, Onset)*.

Event initiations identified by definitions with placeholders are handled similarly. Suppose the initiation event is tagged by a *(Def/XXX/YYY, Onset)* where *XXX* is a defined name and *YYY* is the value substituted for the '#' placeholder. The end of this event's temporal scope is marked either by *(Def/XXX/YYY, Offset)* or by another *(Def/XXX/YYY, Onset)*. A subsequent *(Def/XXX/ZZZ, Onset)* where *YYY* and *ZZZ* are different is treated as a completely distinct temporal event.

Table 5.3 summarizes *Onset* and *Offset* usage.

Syntax summary for *Onset* and *Offset*.

Short forms: *(Def/XXX, Onset, (tag-group))*

(Def/XXX/#, Onset, (tag-group))

(Def/XXX, Offset)

Long forms: *(Property/Organizational-property/Def/XXX,Property/Data-property/Data-marker/Temporal-marker/Onset, (tag-group)*

(Property/Organizational-property/Def/XXX/#,Property/Data-property/Data-marker/Temporal-marker/Onset, (tag-group)

Property/Data-property/Data-marker/Temporal-marker/Offset

Notes:

1. *XXX* is the name of the definition.
2. The *(tag-group)* is optional.
3. The additional tag-group is only in effect for that particular scoped event and not for all *XXX*.
4. If the *Def/XXX/#* form is used, the # must be replaced by an actual value.
5. The entire definition identifier *Def/XXX/#*, including the value substituted for the #, is used as the anchor for temporal scope.

For example, the *PlayMovie* definition of the previous section just defines the playing of a movie clip on the screen. The *(tag-group)* might include tags identifying which clip is playing in this instance. This syntax allows one definition name to be used to represent the playing of different clips.

Example: The playing of a Star Wars clip using *PlayMovie*.

Short form:

[event 1]*Sensory-event, (Def/PlayMovie, Onset, (Label/StarWars, (Media-clip, ID/3284)))*

```
.... [The Star Wars movie clip is playing] ....
```

[event n]*Sensory-event, (Def/PlayMovie, Offset)*

Long form:

[event 1]*Event/Sensory-event,(Attribute/Informational/Def/PlayMovie,Data-property/Data-marker/Temporal-marker/Onset,(Attribute/Informational/Label/StarWars,(Item/Object/Man-made-object/Media/Media-clip,Properity/Informational-property/ID/3284)))*

```
.... [The Star Wars movie clip is playing] ....
```

[event n]*Event/Sensory-event,(Attribute/Informational/Def/PlayMovie,Data-property/Data-marker/Temporal-marker/Offset)*

The *PlayMovie* scoped event type can be reused to annotate the playing of other movie clips. However, scoped events with the same defined name (e.g., *PlayMovie*) cannot be nested. The temporal scope of a *PlayMovie* event ends with a *PlayMovie* offset or with the onset of another *PlayMovie* event.

In the previous example, the *PlayMovie* defined name "anchors" the temporal scope, and the appearance of another *Def/PlayMovie* indicates the previous movie has ceased. The *Label* tag identifies the particular movie but does not affect the *Onset*/*Offset* determination.

If you want to have interleaved movies playing, use definitions with placeholder values as shown in the next example. The example assumes a definition *Definition/MyPlayMovie/#* exists.

Example: The interleaved playing of Star Wars and Forrest Gump.

Short form:

[event 1]*Sensory-event, (Def/MyPlayMovie/StarWars, Onset, (Media-clip, ID/3284))*

```
.... [The Star Wars movie clip is playing] ....
```

[event n1] *Sensory-event, (Def/MyPlayMovie/ForrestGump, Onset, (Media-clip, ID/5291))*

```
.... [Both Star Wars and Forrest Gump are playing] ....
```

[event n2]*Sensory-event, (Def/MyPlayMovie/StarWars, Offset)*

```
.... [Just Forrest Gump is playing] ....
```

[event n3]*Sensory-event, (Def/MyPlayMovie/ForrestGump, Offset)*

Because tools need to have the definitions in hand when fully expanding during validation and analysis, tools must gather applicable definitions before final processing. Library functions in Python, Matlab, and JavaScript are being developed to support gathering of definitions and the expansion.

5.3.2 5.3.2. *Duration*

The *Duration* tag is an alternative method for specifying an event with temporal scope. The start of the temporal scope is the event in which the *Duration* tag appears. The end of the temporal scope is implicit and may not coincide with an actual event appearing in the recording. Instead, tools calculate when the scope ends in the data recording.

Duration tags do not need a defined label. *Duration* may be grouped with tags representing the additional information associated with the temporal scope of that event. This grouping usually does not include tags from the *Event* rooted tree.

Example: Use *Duration* for the playing of a 2-s movie clip of Star Wars.

Short form:

Sensory-event,(Duration/2 s, Visual-presentation, (Movie, Label/StarWars), Computer-screen)

Long form:

Event/Sensory-event,(Property/Data-value/Spatiotemporal-value/Temporal-value/Duration/2 s,Property/Sensory-property/Sensory-presentation/Visual-presentation,(Item/Object/Man-made-object/Media/Visualization/Movie,Property/Informational-property/Label/StarWars),Item/Object/Man-made-object/Device/IO-device/Output-device/Display-device/Computer-screen,Property/Informational-property/Description/Play a movie clip for 2 s.)

The *Duration* tag is convenient because its use does not require a *Definition*. The *Duration* tag has the same effect on event context as the onset/offset mechanism explained in Section 5.1: *Onset* and *Offset*. However, the ending time point of events whose temporal scope is defined with *Duration* is not marked by an explicit event in the data recording. This has distinct disadvantages for analysis if the offset is expected to elicit a neural response, which is the case for most events involving visual or auditory presentations.

5.3.3 5.3.3. Temporal offsets with *Delay*

The *Delay* tag is grouped with a set of tags to indicate that the associated tag-group is actually an implicit event that occurs at a time offset from the current event. A typical use case is when the user response time to a stimulus is recorded as a delay time relative to the onset of the corresponding stimulus event. This strategy is convenient for some time-locked analyses. HED tools could be developed to support the expansion of delayed events into actual events in the event stream, provided delays were consistently provided as signed numerical values relative to the anchor onset.

In the following example, a trial consists of the presentation of a cross in the center of the screen. The participant responds with a button press upon seeing the cross. The response time of the button push is recorded relative to the stimulus presentation as part of the stimulus event.

Example: Use *Delay* for offset events.

Short form:

Sensory-event, Experimental-stimulus, Visual-presentation,(Cross, (Center-of, Computer-screen)),(Agent-action, Delay/2.83 ms, (Participant-response, (Press, Mouse-button)))

Long form:

Event/Sensory-event,Property/Task-property/Task-event-role/Experimental-stimulus,Property/Sensory-property/Sensory-presentation/Visual-presentation,(Item/Object/Geometric-object/2D-shape/Cross,(Relation/Spatial-relation/Center-of,Item/Object/Man-made-object/Device/IO-device/Output-device/Display-device/Computer-screen)),(Event/Agent-action,Property/Data-property/Data-value/Spatiotemporal-value/Temporal-value/Delay/2.83 ms,(Property/Task-property/Task-event-role/Participant-response,(Action/Move/Move-body-part/Move-upper-extremity/Press/,Item/Object/Man-made-object/Device/IO-device/Input-device/Computer-mouse/Mouse-button))),Property/Informational-property/Description/A cross is displayedin the center of the screen and the participant responds by pushing a button.

Notice that the *Agent-action* tag from the *Event* subtree is included in the *Delay* tag-group. This allows tools to identify this tag-group as representing a distinct event. For BIDS datasets, such response delays would be in value columns of the `_events.tsv` event files. The HED annotation for the JSON sidecar corresponding to these files would contain a `#`. At HED expansion time, tools replace the # with the column value (2.83) corresponding to each event.

5.4 5.4. Event streams

An event stream is a sequence of events in a data recording. The most obvious event stream is the sequence consisting of all the events in the recording, but there are many other possible streams such as the stream consisting of all sensory events or the stream consisting of all participant response events.

Event streams can be identified and tagged using the *Event-stream* tag, allowing annotators to more easily identify subsets of events and interrelationships of events within those event sequences. An event having the tag *Event-stream/XXX* is part of event stream XXX.

Example: Tag a face event as part of the *Face-stream* event stream.

Short form:

Sensory-event, Event-stream/Face-stream, Visual-presentation, (Image, Face)

Long form:

Event/Sensory-event,Property/Organizational-property/Event-stream/Face-stream,Property/Sensory-property/Sensory-presentation/Visual-presentation,(Item/Object/Man-made-object/Media/Visualization/Image,Item/Biological-item/Anatomical-item/Body-part/Head/Face)

Using a tag to identify an event stream makes it easier for downstream tools to compute relationships among subsets of events.

5.5 5.5. Event context

Event annotations generally focus on describing what happened at the instant an event was initiated. However, the details of the setting in which the event occurs also influence neural responses. For the *PlayMovie* example of the previous section, events that occur between the *Onset* and *Offset* pairs for *PlayMovie* should inherit the information that a particular movie is playing without requiring the user to explicitly enter those tags for every intervening event.

The process of event context mapping should be deferred until analysis time because other events might be added to the event file after the initial annotation of the recording. For example, a user might run a tool to mark blink or other features as events prior to doing other analyses. HED uses the *Event-context* tag to accomplish the required context mapping.

In normal usage, **the *Event-context* tag is not used directly by annotators**. Rather, tools insert the *Event-context* tag at analysis time to handle the implicit context created by enduring or scoped events. However, annotators may use the tag when an event has explicit context information that must be accounted for.

Syntax summary for *Event-context*.

Short form: *(Event-context, other-tags)*

Long form: *(Property/Organizational-property/Event-context, other-tags)*

Notes:

1. An event can have at most one *Event-context*.
2. HED-compliant analysis tools should insert the annotations describing each temporally scoped event into the *Event-context* tag group of the events within its temporal scope during final assembly before analysis of the event.
3. Other task-event relationships may be inserted as tags within the *Event-context* tag group either at annotation time or analysis time.

5.6 5.6. Experimental design

Most experiments are conducted by varying certain aspects of the experiment and measuring the resulting responses while carefully controlling other aspects. The intention of the experiment is annotated using the HED *Condition-variable*, *Control-variable*, and *Indicator-variable* tags.

The *Condition-variable* tag is used to mark the independent variables of the experiment – those aspects of an experiment that are explicitly varied in order to observe an effect or to control bias. Contrasts, a term that appears in the neuroscience and statistical literature, are examples of experimental conditions, as are factors in experimental designs.

The *Indicator-variable* tag is used to mark quantities that are explicitly measured or calculated to evaluate the effect of varying the experimental conditions. Indicator variables often fall into the *Event/Data-feature* category. Sometimes

the values of these data features are explicitly annotated as events. Researchers should provide a sufficiently detailed description of how to compute these data features so that they can be reproduced.

The *Control-variable* tag represents an aspect of the experiment that is held constant throughout the experiment, often to remove variability.

Researchers should use *Condition-variable*, *Control-variable*, and *Indicator-variable* tags to capture the experiment intent and organization in as much detail as possible. Consistent and detailed description allows tools to extract the experiment design from the data in a machine-actionable form. Good tagging processes suggest creating definitions with understandable names to define these aspects of the dataset. This promotes easy searching and extraction for analyses such as regression or other modeling of the experimental design.

To illustrate the use of condition-variables to document experiment design, consider an experiment in which one of the conditions is the rate of presentation of images displayed on the screen. The experiment design compares responses under slow and fast image presentation rate conditions. To avoid unfortunate resonances due to a poor choice of rates, the "slow" and "fast" rate conditions each consist of three possible rates. Selection among the three eligible rates for the given condition is done randomly.

In analysis, the researcher would typically combine the "slow presentation" trials into one group and the "fast presentation" trials into another group even though the exact task condition varies within the group varies according This type of grouping structure is very common in experiment design and can be captured by HED tags in a straightforward manner by defining condition variables for each group and using the # to capture variability within the group.

Example: Condition variables for slow and fast visual presentation rates.

Short form:

(Definition/SlowPresentation/#,(Condition-variable/Presentation, Visual-presentation, Computer-screen, Temporal-rate/#))

(Definition/FastPresentation/#,(Condition-variable/Presentation, Visual-presentation, Computer-screen, Temporal-rate/#))

Long form:

(Property/Informational-property/Definition/SlowPresentation/#,(Property/Organizational-property/Condition-variable/Presentation,Property/Sensory-property/Sensory-presentation/Visual-presentation,Item/Object/Man-made-object/Device/IO-device/Output-device/Display-device/Computer-screen,Property/Data-property/Data-value/Spatiotemporal-value/Rate-of-change/Temporal-rate/#))

(Property/Informational-property/Definition/FastPresentation/#,(Property/Organizational-property/Condition-variable/Presentation,Property/Sensory-property/Sensory-presentation/Visual-presentation,Item/Object/Man-made-object/Device/IO-device/Output-device/Display-device/Computer-screen,Property/Data-property/Data-value/Spatiotemporal-value/Rate-of-change/Temporal-rate/#))

Organizational tags such as *Condition-variable* are often used in the tag-groups of temporally scoped events. The *Onset* of such an event represents the start of the *Condition-variable*. The corresponding *Offset* marks the end of the period during which this condition is in effect. This type of annotation makes it straightforward to extract the experimental design from the events.

Example: Annotation using *SlowPresentation* condition.

Short form:

Sensory-event, (Def/SlowPresentation/1 Hz, Onset)

Long form:

Event/Sensory-event,(Property/Organizational-property/Def/SlowPresentation/1 Hz,Property/Data-property/Data-marker/Temporal-marker/Onset)

During analysis, the *Def* tags will be replaced with the actual definition's tag group with an included *Def-expand* tag giving the definition's name. Note: expansion is done by tools at analysis time.

Example: Expanded form of the previous example.

Short form with expansion:

Sensory-event,((Def-expand/SlowPresentation, Condition-variable/Presentation,Visual-presentation, Computer-screen, Temporal-rate/1 Hz), Onset)

Long form with expansion: Event/Sensory-event,*

((Property/Organizational/Def-expand/SlowPresentation,Property/Organizational/Condition-variable/Presentation,Property/Sensory-property/Sensory-presentation/Visual-presentation,Item/Object/Man-made-object/Device/IO-device/Output-device/Display-device/Computer-screen,Property/Data-property/Data-value/Spatiotemporal-value/Rate-of-change/Temporal-rate/1 Hz),Property/Data-property/Data-marker/Temporal-marker/Onset)

Properly annotated condition variables and response variables can allow researchers to understand the details of the experiment design and perform analyses such as ANOVA (ANalysis Of VAriance) or regression to extract the dependence of responses on the condition variables. The time-organization of an experiment can be annotated with the Organizational tags *Time-block* and *Task-trial* and used for visualizations of experimental layout.

A typical experiment usually consists of a sequence of subject task-related activities interspersed with rest periods and/or off-line activities such as filling in a survey. The *Time-block* tag is used to mark a contiguous portion of the data recording during which some aspect of the experiment conditions is fixed. *Time-block* tags can be used to represent temporal organization in a manner similar to the way *Condition-variable* tags are used to represent factors in an experiment design.

5.7 5.7. Specialized annotation

A significant problem with schema design is term accretion. Each type of experiment will have specific terms or concepts that are important for the experiment's purpose or design but are not widely applicable to other experiments. Schema designers might be tempted to add terms specific to familiar experiments or for annotators to extend the schema tree with terms specific to their experiments during annotation.

The *Parameter* tag and its children *Parameter-label* and *Parameter-value* are general-purpose tags designed to fill the missing term gap. They can be used to tag important specific concepts in a way that can be used for automated tools without triggering problems of accretion. For example, consider the problem of how to annotate repetition lag between successive presentations of a particular face image. There are several ways to annotate, but annotating with *Parameter* is a good compromise between clarity and machine-actionability.

Example: Annotate face repetition and interval using *Parameter-value*.

Short form:

(Parameter-label/Count-of-this-face, Parameter-value/2)(Parameter-label/Face-count-since-this-face-last-shown, Parameter-value/15)

Annotate the number of times a face image has appeared and the interval since last time this face was shown using more specific tags for the value *Parameter-value*:

Example: Annotate the number of times a face image has appeared.

Short form:

(Parameter-label/Count-of-this-face, Item-count/2),(Parameter-label/Face-count-since-this-face-last-shown,Item-count-interval/15),

Long form:

(Property/Informational-property/Parameter/Parameter-label/Count-of-this-face,Property/Data-property/Data-value/Quantitative-value/Item-count/2),(Property/Informational-property/Parameter/Parameter-label/Face-count-since-this-face-last-shownProperty/Data-property/Data-value/Quantitative-value/Item-count-interval/15)

Using more specific tags as in the second version allows downstream tools to treat the value as numeric integers, facilitating automated processing. The use of *Parameter* alerts downstream tools that this entity represents something that annotators regard as important to compute or record for analysis. Summary tools can extract the experimental parameters and their values, while statistical tools can look for dependencies on these variables. The parameter names are designed to be self-documenting. Parameters are often used for derived values such as response times that are used as indicator variables in the experiment. They are also sometimes used as part of control variable definitions.

CHAPTER

SIX

6. INFRASTRUCTURE

This section gives an overview of the HED infrastructure. Additional details and links to specific tools are available in (Tools.md)[Tools.md].

6.1 6.1. Short and long forms

Tools that are third-generation HED-compliant must be able to handle both short-form and long-form versions of HED strings. Analysis tools often need to transform all HED tags to long form before processing. To this end, mapping functions are being developed in Python, Matlab, and JavaScript. These libraries also provide mapping from long form to short form. As illustrated in the previous sections, the short form is much more readable and compact.

6.2 6.2. File formats

Dataset events are often represented using spreadsheets either in `.tsv` or Excel format. The rows of each spreadsheet correspond to events, while the columns contain identifying information pertaining to the events. The first row of each spreadsheet usually contains column names that document what each column represents. Usually one column contains the time of the event. Other columns may contain categorical values, other values, or HED strings.

Categorical column values are chosen from a small, explicitly defined subset. Value columns may contain numeric values or other types of values such as file names. HED tools assume that event files are spreadsheets, either in BIDS (`.tsv`) format or Excel format.

The HED tools require that each column of an event file contains items of the same class (categorical or value) and that value columns contain items of the same basic type. Files not satisfying these requirements may need additional processing before being handled by HED tools.

6.2.1 6.2.1. BIDS event files

BIDS (Brain Imaging Data Structure) is a specification along with supporting tools for organizing and describing brain imaging and behavioral data. BIDS event files satisfy the criteria of the previous section.

BIDS supports HED annotation of events. BIDS events appear in tab-separated value (`_events.tsv`) files in various places in the dataset hierarchy. BIDS event files must have an `onset` column and a `duration` column. The following shows an excerpt from a BIDS event file:

Example: Excerpt from a BIDS event file.

```
onset  duration  trial_type  response_time stim_file
1.2    0.6       go          1.435         images/red_square.jpg
5.6    0.6       stop        1.739         images/blue_square.jpg
```

The `trial_type` column contains categorical values, while the `response_time` and `stim_file` columns contain non-categorical values. In theory `stim_file` could be considered a categorical column if there were just a few possible images, but this would not be common usage. BIDS allows an optional column named `HED` to contain HED strings relevant for the event instance. The above example does not have this column.

Processing tools read these event files and create their own event representation. The Python version of HEDTools uses the Pandas `DataFrame` for its low-level representations. For MATLAB programs, the dataset events are often held in `struct` arrays.In EEGLAB, for example, the events for an EEG data recording appear in the `EEG.event` structure array. The time of the event is given in frames in the `EEG.event.latency` field for data that has not been epoched.

6.2.2 6.2.2. BIDS sidecars

BIDS also recommends data dictionaries in the form of JSON sidecars to document the meaning of the data in the event files.HEDTools assume that dictionaries for event metadata are contained in BIDS-compatible sidecars.

BIDS allows the tagging of both categorical and non-categorical columns in these sidecars as explained in the HED appendix of the BIDS specification. Internally, EEGLAB and CTAGGER use mapping objects that are stored in the EEG structure. However, these mapping options can be written to or read from BIDS JSON sidecars.

Each event file spreadsheet column containing categorical values may also have a categorical dictionary that documents the meaning of the data in that column. HED also provides for the HED tagging of non-categorical columns as explained below. The following example shows the JSON sidecar format for annotating the same event file of the previous section. The `"HED"` key for the `"trial_type"` column indexes the categorical dictionary associated with the `trial_type` column in the event file.

Example: JSON sidecar for annotating the columns of an event file.

```
{
    "trial_type": {
        "LongName": "Event category",
        "Description": "Indicator of type of action that is expected.",
        "Levels": {
            "go": "A red square is displayed to indicate starting",
            "stop": "A blue square is displayed to indicate stopping."
        },
        "HED": {
            "go": "Sensory-event, Visual-presentation, (Square, Red), (Computer-screen,␣
↪Center-of), Description/A red square is displayed to indicate starting.",
            "stop": "Sensory-event, Visual-presentation, (Square, Blue), (Computer-
↪screen, Center-of), Description/A blue square is displayed to indicate stopping."
        }
    },
    "response_time": {
        "LongName": "Response time after stimulus",
        "Description": "Time from stimulus until subject presses button.",
        "Units": "ms",
        "HED": "(Delay/# ms, Agent-action, Experiment-participant, Press, Mouse-button,␣
↪Description/Time from stimulus until subject presses button)"
```

(continues on next page)

(continued from previous page)

```
    },
    "stim_file": {
        "LongName": "Stimulus file name",
        "Description": "Relative path of the stimulus image file",
        "HED": "Pathname/#, Description/Relative path of the stimulus image file"
    }
}
```

Non-categorical columns such as `response_time` and `stim_file` have a dictionary "HED" value consisting of a HED string rather than another dictionary. This HED string must have a single # placeholder. The corresponding value in the spreadsheet column replaces the # when the event annotation is assembled.

6.2.3 6.2.3. HED version in BIDS

The HED version is included as the value of the `"HEDVersion"` key in the `dataset_description.json` metadata file located at the top level in a BIDS dataset. HEDTools retrieve the appropriate HED schema directly from GitHub when needed. The following examples shows how a BIDS user specifies that HED version 8.0.0 is used for a dataset called "A wonderful experiment". BIDS locates the appropriate version of the schema on GitHub and downloads it during the validation process. The following examples shows a simple `dataset_description.json`.

Example: BIDS dataset description using HED version 8.0.0.

```
{
   "Name": "A wonderful experiment",
   "BIDSVersion": "1.4.0",
   "HEDVersion": "8.0.0"
}
```

6.3 6.3. Levels of validation

Validation of HED annotations is an essential step in using HED for large-scale, reproducible analysis. Third-generation HED encourages a more detailed and useful documentation of events and provides mechanisms for mapping the interrelationships of events and task intent. The additional annotation power also requires more extensive validation to assure consistency across annotations. HED-validators are provided in both Python and JavaScript. There is also a MATLAB wrapper for the Python validator functions.

There are five levels of validation: tag level, string level, dictionary level, event level, and data-recording level. Previous generations of HED only required validation at the first four of these. Since third-generation HED can document relationships across events, it also requires an additional level to validate cross-event relationships. Validation can also be categorized as syntactic or semantic. Syntactic validation, which occurs mainly at the HED tag and HED string levels, tests that the tags are properly formed, independently of the HED schema or purpose of the tags. Semantic validation tests that the tags are used correctly and that they comply with the relevant schema. Syntactic validation is usually done initially during the parsing of the HED strings into HED tags.

6.3.1 6.3.1. Tag validation

HED tag level validation checks each individual HED tag against its associated schema. The long-form tag must be in the schema. HED tags that take a value (have a # child in the schema) must have values that only contain appropriate characters. If the HED tag # has a *unitClass* attribute, the units must comply with those of the specified *unitClass*. If the HED tag has additional nodes beyond the leaf node in the schema, the *extensionAllowed* attribute must be in effect for the leaf node.

6.3.2 6.3.2. String validation

HED string level validation focuses on the proper formation of HED strings and tag-groups within the HED strings. Syntactic HED string validation includes matching of parentheses and proper delimiting of HED tags by commas. Semantic HED string validation includes verification that HED definitions have the proper form.

6.3.3 6.3.3. Sidecar validation

HED dictionary validation assumes that the dictionaries have been written in the JSON format of Chapter 6: BIDS sidecars. The validation is similar to HED string evaluation, but the error messages are keyed to dictionary location rather than to line numbers in the event file or spreadsheet. The validator checks that there is exactly one # in the HED string annotation associated with each non-categorical column. The # placeholder should correspond to a # in the HED schema, indicating that the parent tag expects a value. If the placeholder is followed by a unit designator, the validator checks that these units are consistent with the unit class of the corresponding # in the schema. The units are not mandatory.

6.3.4 6.3.4. Event validation

Dataset formats such as BIDS (Brain Imaging Data Structure) allow users to provide HED tags in multiple places. For example, if a study uses local codes to represent different types of events, the dataset specification often allows users to use the local codes when listing the events and then provide some format of dictionary mapping local codes to tags. During event level validation, all of these tags must be assembled into a HED string representing the full HED annotation for that event.

Several tag attributes are validated at this stage. The expanded event-level HED string annotating an event must include all tags with the *required* attribute and only one copy of each tag with the *unique* attribute.

6.3.5 6.3.5. Recording validation

The introduction of definitions and temporal scope has added additional complexity to validation. Instead of being able to validate the HED string for each event individually, third generation HED validators must also check consistency across all events in the data-recording. This validation requires multiple passes through of the assembled HED tags associated with the data-recording.

Since *Definition* tags can appear anywhere in the event annotations for a data recording, an initial scan must be made to assemble all definitions for a data recording and to make sure that the definition names are unique.

To validate temporal scope, the validator must assure that each *Onset* and *Offset* tag is associated with an appropriately defined label. The validator must also check to make sure that *Onset* and *Offset* tags are properly matched within the data recording.

6.4 6.4. Analysis tools

Third-generation HED analysis tools also require some additional infrastructure. These tools should call the HED libraries to fully expand the tags for a data recording before doing analysis. In addition to converting all HED tags to their long form, the library tools can remove the definitions and replace *def/* with the full tag expansion with any defined labels.

Each event that is within the temporal scope of a scoped event, should have the scoped event information added to the *Event-context* tag group of the intermediate event upon request. *Delay* tag expansions as insertions of actual events should also be supported. *Duration* tag expansion in which offset events should be inserted should also be supported. The HED tools should provide this expansion capability as well as a standardized representation of events in a data recording to enable tools to use a standard API for accessing tag information.

6.5 6.5. BIDS support in HED

HED provides a JavaScript validator in the [hed-javascript](https://github. com/hed-standard/hed-javascript) repository, which is available as an installable package via npm. The [BIDS validator](https://github. com/bids-standard/bids-validator) incorporates calls to this package to validate HED tags in BIDS datasets.

The hedtools package includes input functions that use Pandas data frames to construct internal representations of HED-annotated event files. Plans are underway to make this package available on the PyPI package index for easy installation.

CHAPTER

SEVEN

7. LIBRARY SCHEMA

The variety and complexity of events in electrophysiological experiments makes full documentation challenging. As more experiments move out of controlled laboratory environments and into less controlled virtual and real-world settings, the terminology required to adequately describe events has the potential to grow exponentially.

In addition, experiments in any given subfield can contribute to pressure to add overly-specific terms and jargon to the schema hierarchy—for example, adding musical terms to tag events in music-based experiments, video markup terms for experiments involving movie viewing, traffic terms for experiments involving virtual driving, and so forth.

Clinical fields using neuroimaging also have their own specific vocabularies for describing data features of clinical interest (e.g., seizure, sleep stage IV). Including these discipline-specific terms quickly makes the base HED schema unwieldy and less usable by the broader user community.

Third generation HED instead introduces the concept of the **HED library schema**. To use a programming analogy, when programmers write a Python module, the resulting code does not become part of the Python language or core library. Instead, the module becomes part of a library used in conjunction with core modules of the programming language.

Similar to the design principles imposed on function names and subclass organization in software development, HED library schemas must conform to some basic rules:

As in Python programming, we anticipate that many HED schema libraries may be defined and used, in addition to the base HED schema. Libraries allow individual research communities to annotate details of events in experiments designed to answer questions of interest to particular research or clinical communities. Since it would be impossible to avoid naming conflicts across schema libraries that may be built in parallel by different user communities, HED supports schema library namespaces. Users will be able to add library tags qualified with namespace designators. All HED schemas, including library schemas, adhere to semantic versioning.

7.1 7.1. Defining a schema

A HED library schema is defined in the same way as the base HED schema except that it has an additional attribute name-value pair, `library="xxx"` in the schema header. We will use as an illustration a library schema for driving. Syntax details for a library schema are similar to those for the base HED schema. (See *Appendix A: Schema format* for more details).

Example: Driving library schema (MEDIAWIKI template).

```
HED library="driving" version="1.0.0"
!# start schema
   [... contents of the HED driving schema ...]
!# end schema
```

(continues on next page)

(continued from previous page)

```
   [... required sections specifying schema attribute definitions ...]
!# end hed
```

The required sections specifying the schema attributes are *unit-class-specification*, *unit-modifier-specification*, *value-class-specification*, *schema-attribute-specification*, and *property-specification*.

Example: Driving library schema (XML template).

```
<?xml version="1.0" ?>
<HED library="driving" version="1.0.0">
    [... contents of the HED_DRIVE schema ... ]
</HED>
```

The schema XML file should be saved as `HED_driving_1.0.0.xml` to facilitate specification in tools.

7.2 7.2. Schema namespaces

As part of the HED annotation process, users must associate a standard HED schema with their datasets. Users may also include tags from an arbitrary number of additional library schemas. For each library schema used to annotate a data recording, the user must associate a local name with the appropriate library schema name and version. Each library must be associated with a distinct local name within a recording annotations. The local names should be strictly alphabetic with no blanks or punctuation.

The user must pass information about the library schema and their associated local names to processing functions. HED uses a standard method of identifying namespace elements by prefixing HED library schema tags with the associated local names. Tags from different library schemas can be intermixed with those of the base schema. Since the node names within a library must be unique, annotators can use short form as well as fully expanded tag paths for library schema tags as well as those from the base-schema.

Example: Driving library schema example tags.

```
dp:Action/Drive/Change-lanes
dp:Drive/Change-lanes
dp:Change-lanes
```

A colon (`:`) is used to separate the qualifying local name from the remainder of the tag. Notice that *Action* also appears in the standard HED schema. Identical terms may be used in a library schema and the standard HED schema. Use of the same term implies a similar purpose. Library schema developers should try not to reuse terms in the standard schema unless the intention is to convey a close or identical relationship.

7.3 7.3. Attributes and classes

In addition to the specification of tags in the main part of a schema, a HED schema has sections that specify unit classes, unit modifiers, value classes, schema attributes, and properties. The rules for the handling of these sections for a library schema are as follows:

7.3.1 7.3.1. Required sections

The required sections of a library schema are: the *schema-specification*, the *unit-class-specification*, the *unit-modifier-specification*, the *value-class-specification* section, the *schema-attribute-specification* section, and the *property-specification*. The library schema must include all required schema sections even if the content of these sections is empty.

7.3.2 7.3.2. Relation to base schema

Any schema attribute, unit class, unit modifier, value class, or property used in the library schema must be specified in the appropriate section of the library schema regardless of whether these appear in base schema. Validators check the library schema strictly on the basis of its own specification without reference to another schema.

7.3.3 7.3.3. Schema properties

HED only supports the schema properties listed in Table B.2: *boolProperty*, *unitClassProperty*, *unitModifierProperty*, *unitProperty*, and *valueClassProperty*.If the library schema uses one of these in the library schema specification, then its specification must appear in the *property-specification* section of the library schema.

7.3.4 7.3.4. Unit classes

The library schema may define unit classes and units as desired or include unit classes or units from the base schema. Similarly, library schema may define unit modifiers or reuse unit modifiers from the base schema. HED validation and basic analysis tools validate these based strictly on the schema specification and do not use any outside information for these.

7.3.5 7.3.5. Value classes

The standard value classes (*dateTimeClass[]*, nameClass, numericClass[*]*, posixPath[]*, textClass[*]*) if used, should have the same meaning as in the base schema. The hard-coded behavior associated with the starred ([*]) value classes will be the same. Library schema may define additional value classes and specify their allowed characters, but no additional hard-coded behavior will be available in the standard toolset. This does not preclude special-purpose tools from incorporating their own behavior.

7.3.6 7.3.6. Schema attributes

The standard schema attributes (*allowedCharacter*, *defaultUnits*, *extensionAllowed*, *recommended*, *relatedTag*, *requireChild*, *required*, *SIUnit*, *SIUnitModifier*, *SIUnitSymbolModifier*, *suggestedTag*, *tagGroup*, *takesValue*, *topLevelTagGroup*, *unique*, *unitClass*, *unitPrefix*, *unitSymbol*, *valueClass*) should have the same meaning as in the base schema. The hard-coded behavior associated with the schema attributes will be the same. Library schema may define additional schema attributes. They will be checked for syntax, but no additional hard-coded behavior will be available in the standard toolset. This does not preclude special-purpose tools from incorporating their own behavior.

7.3.7 7.3.7. Syntax checking

Regardless of whether a specification is in the base-schema or not, HED tools can perform basic syntax checking.

Basic syntax checking for library schema.

1. All attributes used in the schema proper must be defined in the schema attribute section of the schema.
2. Undefined attributes cause an error in schema validation.
3. Similar rules apply to unit classes, unit modifiers, value classes, and properties.
4. Actual handling of the semantics by HED tools only occurs for entities appearing in the base schema.

7.4 7.4. library schemas in BIDS

The most common use case (for 99.9% of the HED users) is to use one of the standard HED schemas available on GitHub in the `hedxml` directory of the `hed-specification` repository (https://github.com/hed-standard/hed-specification/tree/master/hedxml).

This section explains the changes that are being proposed in BIDS to accommodate access to HED library schemas. This section will be updated as the proposals progress though the BIDS review process. All `"fileName"` keys in the following discussion point to the names of files located in the `./code` directory of the dataset tree.

The major change to the BIDS specification is to allow the value associated with the `"HEDVersion"` key in the `dataset_description.json` file to be a dictionary rather than a string expressing the HED version. This proposed change will allow users more flexibility in specifying the base HED schema and will accommodate an arbitrary number of library schemas. The allowed top-level keys in this dictionary are: `"version"`, `"fileName"`, and `"libraries"`. The `"version"` and `"fileName"` keys pertain to the base HED schema, and if both are specified, `"version"` always takes precedence.

The `"libraries"` key points to a library dictionary. The keys of the library dictionary are the nicknames used in the dataset to reference the schema. The values The `"version"` key specifies the HED version number of a schema in the standard library and has the same effect as directly specifying. The following example specifies a base HED schema located in the `./code/myLocalSchema.xml` file of the dataset and two library schemas with nicknames `"la"` and `"lb"`. `

Example: Proposed specification of library schema in BIDS.

```
{
    "Name": "A wonderful experiment",
    "BIDSVersion": "1.4.0",
    "HEDVersion": {
```

(continues on next page)

(continued from previous page)

```
        "path": "mylocalSchema.xml",
        "libraries": {
            "la": {
                "libraryName": "libraryA",
                "version": "1.0.2"
            },
            "lb": {
                "path": "HED_libraryB_0.5.3.xml"
            }
        }
    }
}
```

The "la" library schema is the ./hedxml/HED_libraryA_1.0.2.xml file found in the hed-schema-library repository on the hed-standard working group GitHub site. HED tags from this library have the la: prefix (e.g., la:XXX). The "lb" library schema can be found in the ./code/HED_libraryB_0.5.3.xml file in the BIDS dataset. Tags from this library are prefixed with lb:. NOTE: This specification is preliminary and is waiting the resolution of BIDS formats for specifying external files as outline in BIDS specification PR #820.

CHAPTER

EIGHT

A. SCHEMA FORMAT

HED schema developers generally do initial development of the schema using `.mediawiki` format. The tools to convert schema between `.mediawiki` and `.xml` format are located in the `hed.schema` module of the hedtools project of the hed-python repository located athttps://github.com/hed-standard/hed-python. All conversions are performed by converting the schema to a `HedSchema` object. Then modules `wiki2xml.py` and `xml2wiki.py` provide top-level functions to perform these conversions. This appendix presents the rules for HED base and library schema in `.mediawiki` and `.xml` formats.

8.1 A.1. Mediawiki file format

The rules for creating a valid `.mediawiki` specification of a HED schema are given below. The format is line-oriented, meaning that all information about an individual entity should be on a single line. Empty lines and lines containing only blanks are ignored.

8.1.1 A.1.1. Overall file layout

Overall layout of a HED MEDIAWIKI schema file.

```
header-line
prologue
            . . .
!# start schema
schema-specification
!# end schema
unit-class-specification
unit-modifier-specification
value-class-specification
schema-attribute-specification
property-specification
!# end hed
epilogue
```

8.1.2 A.1.2. The *header-line*

The first line of the `.mediawiki` file should be a *header-line* that starts with the keyword `HED` followed by a blank-separated list of name-value pairs.

Table 1: Allowed HED schema header parameters

Name	Level	Description
library	optional	Name of library used in XML file names. The value should only have alphabetic characters.
version	required	A valid semantic version number of the schema.
xmlns	optional	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance".
xsi	optional	xsi:noNamespaceSchemaLocation points to XSD file.

The `xsi` attribute is required if `xmlns:xsi` is given. The `library` and `version` values are used to form the official xml file name and appear as attributes in the <HED> root of the `.xml` file. The versions of the schema that use XSD validation to verify the format (versions 8.0.0 and above) have `xmlns:xsi` and `xsi:noNamespaceSchemaLocation` attributes.

Example: Version 8.0.0 of the HED MEDIAWIKI schema.

```
HED version="8.0.0"
```

The version line must be the first line of the `.mediawiki` file. The schema `.mediawiki` file is `HED-schema-8.0.0.mediawiki` found in https://github.com/hed-standard/hed-specification/tree/master/hedwiki.

Example: Version 8.0.0 of the HED XML base schema.

```
<HED version="8.0.0">
```

The file name in `hedxml` in `hed-specification` is `HED8.0.0.xml`.

Example: Version 1.0.2 of HED test library in MEDIAWIKI format.

```
HED library="test" version="1.0.2"
```

The resulting XML root is:

Example: Version 1.0.2 of HED test library schema in XML format.

```
<HED library="test" version="1.0.2">
```

The file name in `hedxml` in the HED schema library `test` is `HED_test_1.0.2.xml`.

Unknown header-line attributes are translated as attributes of the `HED` root node of the `.xml` version, but a warning is used when the `.mediawiki` file is validated.

8.1.3 A.1.3. Schema section

The beginning of the HED specification is marked by the *start-line*:

```
!# start schema
```

An arbitrary number of lines of informational text can be placed between the header-line and the start-line. Older versions of HED have a CHANGE_LOG as well as information about the syntax and rules. New versions of HED generate a separate change log file for released versions.

The end of the main HED-specification is marked by the end-line:

```
!# end schema
```

The section separator lines (`!# start schema`, `!# end schema`, `!# end hed`) must only appear once in the file and must appear in that order within the file. A section separator is a line starting with `!#`.

The body of the HED specification consists of two types of lines: top-level node-specification specifications and other node specifications. Each specification is a single line in the `.mediawiki` file. Empty lines or lines containing only blanks are ignored. The basic format for a node-specification is:

```
node-name  <nowiki>{attributes}[description]</nowiki>
```

Top-level node names are enclosed in triple single quotes (e.g., `'''Event'''`), while other node names have at least one preceding asterisk (*) followed by a blank and then the name. The number of asterisks indicates the level of the node in the subtree. HED-3G node names can only contain alphanumeric characters, hyphens, and under-bars. They cannot contain blanks and must be unique. HED (2G) and earlier versions allow blanks. Everything after the node name must be contained within <nowiki></nowiki> tags. Placeholder nodes have an empty node name, but are followed by a # enclosed in <nowiki></nowiki> tags.

Example: Different types of HED node specifications.

Top-level:

```
'''Property''' <nowiki>{extensionAllowed} [Subtree of properties.]</nowiki>
```

Normal-level:

```
***** Duration <nowiki>{requireChild} [Time extent of something.]</nowiki>
```

Placeholder-level:

```
****** <nowiki># {takesValue, unitClass=time,valueClass=numericClass}</nowiki>
```

The *Duration* tag of this example is at the fifth level below the root of its subtree. The tag: *Property/Data-property/Data-value/Spatiotemporal-value/Temporal-value/Duration* is the long form. The placeholder in the example is the node directly below *Duration* in the hierarchy.

8.1.4 A.1.4. Other sections

After the line marking the end of the schema (`!# end schema`), the `.mediawiki` file contains the unit class specifications, unit modifier specifications, value class specification, the schema attribute specifications, and property specifications. All of these sections are required starting with HED version 8.0.0 and must be given in this order.

Unit classes specify the kind of units are allowed to be used with a value that is provided for a # value. The unit class specification section starts with `'''Unit classes'''` and lists the type of unit at the first level and the specific units at the second level.

Example: Part of the HED unit class specification for time.

```
'''Unit classes'''
* time <nowiki>{defaultUnits=s}</nowiki>
** second <nowiki>{SIUnit}</nowiki>
** s <nowiki>{SIUnit, unitSymbol}</nowiki>
```

The unit classes can be modified by SI (International System Units) sub-multiples and super-multiples. All unit modifiers are at level 1 of the `.mediawiki` file. Unit modifiers have either the `SIUnitModifer` or the `SIUnitSymbolModifer` to indicate whether they are regular modifiers or symbol modifiers.

Example: Part of the HED unit modifier specification.

```
'''Unit modifiers'''
* deca <nowiki>{SIUnitModifier} [SI unit multiple for 10^1]</nowiki>
* da <nowiki>{SIUnitSymbolModifier} [SI unit multiple for 10^1]</nowiki>
```

Units that have the `SIUnit` attribute can be modified by any unit modifier that has the `SIUnitModifier`. So for example, `second` and `decasecond` are valid time units as are `seconds` and `decaseconds`. Similarly, units that have the `SIUnit` and `unitSymbol` modifiers can be modified with unit modifiers that have the `SIUnitSymbolModifier` attribute.

Value attributes give rules about what kind of value is allowed to be substituted for # placeholder tags.

Example: Part of the HED value class specification.

```
'''Value classes'''
* posixPath <nowiki>{allowedCharacter=/,allowedCharacter=:}[Posix path specification.]</
↪nowiki>
```

The schema attributes specify other characteristics about how particular tags may be used in annotation. These attributes allow validators and other tools to process tag strings based on the HED schema specification, thus avoiding hard-coding particular behavior.

Example: Part of the HED schema attribute specification.

```
'''Schema attributes'''
* allowedCharacter <nowiki>{valueClassProperty}[Attribute of value classes specifying a␣
↪special character that is allowed in expressing the value of a placeholder.]</nowiki>
* defaultUnits <nowiki>{unitClassProperty}[Attribute of unit classes specifying the␣
↪default units for a tag.]</nowiki>
```

(continues on next page)

(continued from previous page)

Notice that in the above example, the schema attributes, themselves have attributes referred to as *HED schema properties*. These schema properties are listed in the `Properties` section of the schema.

Example: Part of the HED schema property specification.

```
'''Properties'''
* valueClassProperty <nowiki>[Attribute is meant to be applied to value classes.]</
↪nowiki>
```

8.2 A.2. XML file format

The XML schema file format has a header, prologue, main schema, definitions, and epilogue sections. The general layout is as follows:

XML layout of the HED schema.

```
<?xml version="1.0" ?>
<HED library="test" version="0.0.1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance
↪" xsi:noNamespaceSchemaLocation="https://github.com/hed-standard/hed-specification/raw/
↪master/hedxml/HED8.0.0-beta.3.xsd">
<prologue>unique optional text blob</prologue>
<schema>
         ...  schema specification  ...
</schema>
<unitClassDefinitions>
   <unitClassDefinition> ... </unitClassDefinition>
                         ...
   <unitClassDefinition> ... </unitClassDefinition>
</unitClassDefinitions>
<unitModifierDefinitions>
   <unitModifierDefinition> ... </unitModifierDefinition>
                            ...
   <unitModifierDefinition> ... </unitModifierDefinition>
</unitModifierDefinitions>

<valueClassDefinitions>
    <valueClassDefinition> ... </valueClassDefinition>
                           ...
    <valueClassDefinition> ... </valueClassDefinition>
</valueClassDefinitions>

<schemaAttributeDefinitions>
   <schemaAttributeDefinition> ... </schemaAttributeDefinition>
                               ...
   <schemaAttributeDefinition> ... </schemaAttributeDefinition>
</schemaAttributeDefinitions>
```

(continues on next page)

(continued from previous page)

```
<propertyDefinitions>
    <propertyDefinition> ... </propertyDefinition>
                            ...
    <propertyDefinition> ... </propertyDefinition>
</propertyDefinitions>

<epilogue>unique optional text blob</epilogue>
</HED>
```

The <prologue>xxx</prologue> and <epilogue>xxx</epilogue> elements are meant to be treated as opaque as far as schema processing goes. In earlier versions of HED the prologue section contained a Change Log for the schema as well as some basic documentation of syntax. The epilogue section contained additional metadata to be ignored during processing. The following subsections give a more detailed description of the format of these sections.

8.2.1 A.2.1. The schema section

The schema section of the HED XML document consists of an arbitrary number of <node></node> elements enclosed in a single <schema></schema> element.

Top-level XML layout of the HED schema.

```
<schema>
    <node> ... </node>
           ...
    <node> ... </node>
</schema>
```

A <node> element contains a required <name> child element, an optional <description> child element, and an optional number of additional <attribute> child elements:

XML layout HED node element.

```
<node>
    <name>xxx</name>
    <description>yyy</description>
    <attribute> ... </attribute>
    <attribute> ... </attribute>
    <attribute> ... </attribute>
    <node> ... <node>
</node>
```

The <name> element text must conform to the rules for naming HED schema nodes. It corresponds to the *node-name* in the mediawiki specification and must not be empty. A # value is used to represent value place-holder elements.

The <description> element has the text contained in the square brackets [] in the .mediawiki node specification. If the .mediawiki specification is missing or has an empty [], the <description> element is omitted.

The optional <attribute> elements are derived from the attribute list contained in curly braces {} of the .mediawiki specification. An <attribute> element has a single non-empty <name></name> child element whose text value corresponds to the node-name of attribute in the corresponding .mediawiki file. If the attribute does not have the boolProperty, then the <attribute> element should also have one or more child <value></value> elements giving the value(s) of the attribute. **Example:** The requireChild attribute represents a boolean value. In the .mediawiki representation this attribute appears as {requireChild} if present and is omitted if absent.

The requireChild attribute represents a boolean value.

Old xml if true:

```
<node requireChild="true"><name>xxx</name></node>
```

New xml if true:

```
<node>
    <name>xxx</name>
    <attribute>
       <name>requireChild</name>
    </attribute>
</node>
```

The suggestedTag attribute has a valid HED tag value. In the mediawiki representation this attribute is omitted if absent and appears when present as shown in this example.

The suggestedTag attribute has a valid HED tag value.

```
{suggestedTag=Sweet,suggestedTag=Gustatory-attribute/Salty}
```

The suggestedTag attribute is meant to be used by tagging tools to suggest additional tags that a user might want to include. Notice that the suggestedTag values are valid HED tags in any form (short, long, or intermediate).

The suggestedTag old format.

Old xml if present:

```
<node suggestedTag="Sweet,Gustatory-attribute/Salty">
    <name>xxx</name>
</node>
```

New xml if present:

```
<node>
   <name>xxx</name>
   <attribute>
      <name>suggestedTag</name>
             <value>Sweet</value>
             <value>Gustatory-attribute/Salty</value>
   </attribute>
</node>
```

8.2.2 A.2.2. Unit classes

The valid HED-3G unit classes are defined in the <unitClassDefinitions> section of the XML schema file, and valid HED-3G unit modifiers are defined in the <unitModifierDefinitions> section. These sections follow a format similar to the <node> element in the <schema> section:

XML layout of the unit class definitions.

```
<unitClassDefinitions>
    <unitClassDefinition> ... </unitClassDefinition>
                          ...
    <unitClassDefinition> ... </unitClassDefinition>
</unitClassDefinitions>
```

The <unitClassDefinition> elements have a required <name>, an optional <description> and an arbitrary number of additional <attribute> child elements. These <attribute> elements describe properties of the unit class rather than of individual unit types. In addition, <unitClassDefinition> elements may have an arbitrary number of <unit> child elements.

XML layout of the unit class definitions.

```
<unitClassDefinition>
    <name>time</name>
    <description>Temporal values except date and time of day.</description>
    <attribute>
       <name>defaultUnits</name>
       <value>s</value>
    </attribute>
    <unit>
       <name>second</name>
       <description>SI unit second.</description>
       <attribute>
          <name>SIUnit</name>
       </attribute>
    </unit>
    <unit>
       <name>s</name>
       <description>SI unit second in abbreviated form.</description>
       <attribute>
          <name>SIUnit</name>
       </attribute>
       <attribute>
          <name>unitSymbol</name>
       </attribute>
    </unit>
</unitClassDefinition>
```

8.2.3 A.2.3. Value classes

Value classes are defined in the <valueClassDefinitions> section of the XML schema file. These sections follow a format similar to the <node> element in the <schema>:

XML layout of the unit class definitions.

```
<valueClassDefinitions>
    <valueClassDefinition> ... </valueClassDefinition>
                           ...
    <valueClassDefinition> ... </valueClassDefinition>
</valueClassDefinitions>
```

8.2.4 A.2.4. Schema attributes

The <schemaAttributeDefinitions> section specifies the allowed attributes of the other elements including the <node>, <unitClassDefinition>, <unitModifierDefinition>, and <valueClassDefinition> elements. The specifications of individual attributes are given in <schemaAttributeDefinition> elements.

XML layout of the schema attribute definitions.

```
<schemaAttributeDefinitions>
    <schemaAttributeDefinition> ...</schemaAttributeDefinition>
                                ...
    <schemaAttributeDefinition> ... </schemaAttributeDefinition>
</schemaAttributeDefinitions>
```

The individual <schemaAttributeDefinition> elements have the following format:

XML layout of the schema attribute definitions.

```
<schemaAttributeDefinition>
    <name>allowedCharacter</name>
    <description>An attribute of value classes indicating a special character that is␣
↪allowed in expressing the value of that placeholder.</description>
    <property>
        <name>valueClassProperty</name>
    </property>
</schemaAttributeDefinition>
```

8.3 A.3. Schema sections

This section gives information about how the various auxiliary sections of the HED schema are used to specify the behavior of the schema elements.

8.3.1 A.3.1. Schema properties

The `property` elements indicate where various schema attributes apply. Their meanings are hard-coded into the schema processors. The following is a list of schema attribute properties.

Table 2: Summary of unit classes and units in HED 8.0.0 (* indicates unit symbol).

Property	Description
boolProperty	Indicates schema attribute values are either true or false.
unitClassProperty	Indicates schema attribute only applies to unit classes.
unitModifierProperty	Indicates schema attribute only applies to unit modifiers.
valueClassProperty	Indicates the schema attribute only applies to value classes.
textClass	Alphanumeric characters, blank, +, -, :, ;, ., /, (,), ?, *, %, $, @, ^, _

Notes on rules for allowed characters in the HED schema.

1. Schema attributes with the `boolProperty` have a <name> node but no <value> node in the XML. Presence indicates true.
2. Schema attributes with the `boolProperty` have both <name> and <value> nodes in the XML.

A given schema attribute can only apply to one type of element (`node`, `unitClassDefinition`, `unitModifierDefinition` or `unit`). Attributes that don't have one of `unitClassProperty`, `unitClassProperty` or `unitProperty` are assumed to apply to `node` elements.

8.3.2 A.3.2. Schema attributes

As mentioned in the previous section schema attributes can only apply to one type of element in the schema as indicated by their property values. Tools hardcode processing based on the schema attribute name. Only the schema attributes listed in the following table can be handled by current HED tools.

Table 3: Schema attributes (* indicates attribute has a value).

Attribute	Target	Description
allowedCharacter*	valueClass	Specifies a character used in values of this class.
defaultUnits*	unitClass	Specifies units to use if placeholder value has no units.
extensionAllowed	node	A tag can have unlimited levels of child nodes added.
recommended	node	Event-level HED strings should include this tag.
relatedTag*	node	A HED tag closely related to this HED tag.
requireChild	node	A child of this node must be included in the HED tag.
required	node	Event-level HED string must include this tag.
SIUnit	unit	This unit represents an SI unit and can be modified.
SIUnitModifier	unitModifier	Modifier applies to base units.
SIUnitSymbolModifier	unitModifier	Modifier applies to unit symbols.
suggestedTag*	node	Tag could be included with this HED tag.
tagGroup	node	Tag can only appear inside a tag group.
takesValue	node #	Placeholder (#)should be replaced by a value.
topLevelTagGroup	node	Tag (or its descendants) can be in a top-level tag group.
unique	node	Tag or its descendants can only occur once in an event-level HED string.
unitClass*	node #	Unit class this replacement value belongs to.
unitPrefix	unit	Unit is a prefix (e.g., $ in the currency units).
unitSymbol	unit	Tag is an abbreviation representing a unit.
valueClass*	node #	Type of value this is.

Normally the allowed characters are listed individually as values of the `allowedCharacter` attribute. However, the word `letters` designates upper and lower case alphabetic characters are allowed. Further, the word `blank` indicates a space is an allowed character, and the word `digits` indicates the digits 0-9 may be used in the value.

If placeholder (`#`) has a `unitClass`, but the replacement value for the placeholder does not have units, tools use the value of `defaultUnits` if the unit class has them. For example, the `timeUnits` has the attribute `defaultUnits=s` in HED 8.0.0. The tag `Duration/3` is assumed to be equivalent to `Duration/3 s` because `Duration` has `defaultUnits` of `s`.

The `extensionAllowed` tag indicates that descendents of this node may be extended by annotators. However, any tag that has a placeholder (`#`) child cannot be extended, regardless of `extensionAllowed` since it single child is always interpreted as a user-supplied value.

Tags with the 'required' or 'unique' attributes cannot appear in definitions.

In addition to the attributes listed above, some schema attributes have been deprecated and are no longer supported in HED, although they are still present in earlier versions of the schema. The following table lists these.

Table 4: Schema attributes (* indicates attribute has a value).

Schema attribute	Target	Description
default	node #	Indicates a default value used if no value is provided.
position	node	Indicates where this tag should appear during display.
predicateType	node	Indicates the relationship of the node to its parent.

The `default` attribute was not implemented in existing tools. The attribute is not used in HED-3G. Only the `defaultUnits` for the unit class will be implemented going forward.

The `position` attribute was used to assist annotation tools, which sought to display required and recommend tags before others. The position attribute value is an integer and the order can start at 0 or 1. Required or recommended tags without this attribute or with negative position will be shown after the others in canonical ordering. The tagging strategy of HED-3G using decomposition and definitions. The `position` attribute is not used for HED-3G.

The predicateType attribute was introduced in HED-2G to facilitate mapping to OWL or RDF. It was needed because the HED-2G schema had a mixture of children that were properties and subclasses. The possible values of predicateType were propertyOf, subclassOf, or passThrough to indicate which role each child node had with respect to its parent. The schema vocabulary redesign of HED-3G eliminated this issue. The attribute is ignored by tools.

8.3.3 A.3.3. Value classes

HED has very strict rules about what characters are allowed in various elements of the HED schema, HED tags and the substitutions made for # placeholders. These rules are encoded in the schema using value classes. When a node name or placeholder substitution is given a particular value class, that name or substituted value can only contain the characters allowed for that value class.

The allowed characters for a value class are specified in the definition of each value class. The HED validator and other HED tools may hardcode information about behavior of certain value classes (for example the numericClass value class). **HED does not allow commas or single quotes in any of its values.**

Table 5: Rules for value classes.

Value class	Allowed characters
dateTimeClass	digits, T, :, -
nameClass	alphabetic characters, -
numericClass	digits, ., -, +, E, e
posixPath	As yet unspecified
textClass	Alphanumeric characters, blank, +, -, :, ;, ., /, (,), ?, *, %, $, @, ^, _

Notes on rules for allowed characters in the HED schema.

1. Commas are not allowed in any values.
2. Date-times should conform to ISO8601 date-time format YYYY-MM-DDThh:mm:ss.
3. Any variation on the full form of ISO8601 date-time is allowed.
4. The name class is for schema nodes and labels.
5. Values that have a value class of numericClass must be valid fixed point of floating point values.
6. Scientific notation is supported with the numericClass.
7. The text class is for descriptions, mainly for use with the *Description/* tag.
8. The posix path class is yet unspecified and currently allows any characters besides commas.

8.3.4 A.3.4. HED unit classes

Unit classes allow annotators to express the units of values in a consistent way. The plurals of the various units are not explicitly listed, but are allowed as HED tools uses standard pluralize functions to expand the list of allowed units. However, Unit symbols represent abbreviated versions of units and cannot be pluralized.

Nodes with the SIUnit modifier may be prefixed with multiple or sub-multiple modifiers. If the SI unit does not also have the unitSymbol attribute, then multiples and sub-multiples with the SIUnitModifier attribute are used for the expansion. On the other hand, units with both SIUnit and SIUnitModifier attributes are expanded using multiples and sub-multiples having the SIUnitSymbolModifier attribute.

Note that some units such as byte are designated as SI units, although they are not part of the standard.

Table 6: Unit classes and units in HED 8.0.0 (* indicates unit symbol).

Unit class	Default units	Units
accelerationUnits	m-per-s^2	m-per-s^2*
angleUnits	rad	radian, rad*, degree
areaUnits	m^2	metre^2, m^2*
currencyUnits	$	dollar, $, point
frequencyUnits	Hz	hertz, Hz*
intensityUnits	dB	dB, candela, cd*
jerkUnits	m-per-s^3	m-per-s^3*
memorySizeUnits	B	byte, B
physicalLength	m	metre, m*, inch, foot, mile
speedUnits	m-per-s	m-per-s*, mph, kph
timeUnits	s	second, s*, day, minute, hour
volumeUnits	m^3	metre^3, m^3*
weightUnits	g	gram, g*, pound, lb

8.3.5 A.3.5. HED unit modifiers

The unit modifiers are can be applied to SI base units to indicate multiples or sub-multiples of the unit. Unit symbols are modified by unit symbol modifiers, whereas non symbol SI units are modified by unit modifiers.

Table 7: Unit modifiers (* indicates unit symbol modifier).

Schema attribute	Description
deca, da*	SI unit multiple representing 10^1
hecto, h*	SI unit multiple representing 10^2
kilo, k*	SI unit multiple representing 10^3
mega, M*	SI unit multiple representing 10^6
giga, G*	SI unit multiple representing 10^9
tera, T*	SI unit multiple representing 10^12
peta, P*	SI unit multiple representing 10^15
exa, E*	SI unit multiple representing 10^18
zetta, Z*	SI unit multiple representing 10^21
yotta, Y*	SI unit multiple representing 10^24
deci, d*	SI unit submultiple representing 10^1
centi, c*	SI unit submultiple representing 10^2
milli, m*	SI unit submultiple representing 10^3
micro, u*	SI unit submultiple representing 10^6
nano, n*	SI unit submultiple representing 10^9
pico, p*	SI unit submultiple representing 10^12
femto, f*	SI unit submultiple representing 10^15
atto, a*	SI unit submultiple representing 10^18
zepto, z*	SI unit submultiple representing 10^21
yocto, y*	SI unit submultiple representing 10^24

CHAPTER

NINE

B. HED ERRORS

This appendix summarizes the error codes used by HED validators and other tools.

HED tools for users (i.e., annotators and analysts) are mainly concerned with HED validation errors relating to incorrectly annotated events.(See *Chapter B.3: HED validation errors* for a listing.) These tools assume that the HED schema are error-free and that schema errors can only occur due to failure to locate or read a HED schema. (See Chapter B.3: Schema loading errors for a listing.)

HED schema developers are mainly concerned with errors and inconsistencies in the schema itself. (See *Chapter B.2: Schema validation errors* for a listing.)

9.1 B.1. HED validation errors

HED_CHARACTER_INVALID: HED string contains an invalid character. HED uses ANSI encoding and does not support UTF-8.

Different parts of a HED string have different rules for acceptable characters as outlined in *Chapter 3.3: Valid characters*.

HED_COMMA_MISSING: HED tag groups and tags must be separated with commas. Commas missing between two HED tags are generally detected as invalid HED tags, rather than as missing commas.

HED_DEF_UNMATCHED: A HED *Def/* label cannot be matched to definition name. A *Def* tag label cannot be correctly matched to a definition name because the definition is missing or defined multiple times.

HED_DEF_VALUE_INVALID: A *Def/* label value is missing or has incorrect format or value. A *Def/* tag value is a schema node name.

A *Def/* tag value does not meet the requirements associated with the placeholder in its definition tag group.

A *Def/* tag has a value, but its corresponding *Definition* does not have a placeholder.

A *Def/* tag does not have a value, but its corresponding *Definition* has a value.

HED_DEFINITION_INVALID: The *Definition* syntax is incorrect or nested. A definition name is invalid or already appears as a schema node.

A definition is not in a top-level tag group.

A definition's enclosing tag group contains another *Definition/* tag.

A definition contains *Def/* or *Def-expand/* tags.

A definition that includes a placeholder (#) does not have exactly two # characters: one after the definition name and one in the definition tag-group body.

A definition has placeholders (#) in incorrect positions.

HED_GENERIC_ERROR: A HED expression raised an uncategorized error. An error that does not fall into other categories.

HED_GENERIC_WARNING: A HED expression raised an uncategorized warning. An warning that does not fall into other categories.

HED_LIBRARY_UNMATCHED: A tag starting with *name:* does not have an associated library. A tag that starts with *name:* is interpreted as a library schema nickname name, but no library schema is defined.

The association of *name* with an actual HED library schema must be passed to the validator when the string containing the tag is validated.

HED_NODE_NAME_EMPTY: An empty tag was detected in a HED string. A tag has extra slashes at beginning, end, or within a tag (implying empty node names).

A HED string starts or ends with a slash.

A HED tag contains consecutive slashes (as this implies a missing term name within a HED tag).

HED_ONSET_OFFSET_ERROR: An *Onset* or *Offset* tag is used incorrectly. An *Onset* or *Offset* tag appears without being grouped with a defined name (using a *Def-expand/* tag group or a *Def/*).

An *Offset* tag appears before an *Onset* tag with the same name (or name/value).

An *Offset* tag of a given name appears after a previous *Offset* tag without the appearance of an intervening *Onset* of the same name.

An *Onset* tag group either lacks an internal tag group or has more than one internal tag group. **Note:** if the *Onset* tag group's definition is in expanded form, the *Def-expand* will be an additional internal tag group.

HED_PARENTHESES_MISMATCH: A HED string has unmatched open and closed parentheses. A HED string does not have the same number of open and closed parentheses.

Open and closed parentheses are not correctly nested.

HED_PLACEHOLDER_INVALID: A # is missing or appears in a place that it should not. A JSON sidecar has a placeholder (#) in the HED dictionary for a categorical column.

A JSON sidecar does not have exactly one placeholder (#) in each HED string representing a value column.

A placeholder (#) is used but its parent in the schema does not have a placeholder child.

HED_REQUIRED_TAG_MISSING: An event-level annotation missing a required tag. An assembled event string does not contain all tags that have the *required* schema attribute.

HED_SIDECAR_KEY_MISSING: (WARNING) A categorical value is missing HED tags in sidecar. The events file column has a HED dictionary in the JSON sidecar but the categorical value does not have a key in the sidecar dictionary.

HED_STYLE_WARNING: (WARNING) An extension or label does not follow HED naming conventions. A tag name does not start with a capital letter with the remainder lower case.

A tag name contains blanks in HED-3G labels or tag extensions. Use hyphens (not under bars) instead.

HED_TAG_EMPTY: Extra commas or empty parentheses indicate empty tags. A HED string has multiple consecutive commas (ignoring white space).

A HED string begins or ends with a comma (which implies an empty HED tag).

A tag group is empty (i.e., empty parentheses are not allowed).

HED_TAG_EXTENDED: (WARNING) HED tag represents an extension from the schema. This tag represents an extension of the HED schema. (Often such tags are really spelling errors and not meant to extend the schema.)

HED_TAG_GROUP_ERROR: A tag does not have its required tag group behavior. A tag has `tagGroup` or `topLevelTagGroup` attribute but is not in an appropriate tag group.

A tag with the `topLevelTagGroup` attribute appears in same tag group as other tags with the `topLevelTagGroup` attribute.

HED_TAG_INVALID: The tag is not valid in this schema. The tag has incorrect format for compliance with this schema.

The tag is used as a tag extension or placeholder value while appearing elsewhere in the schema.

HED_TAG_NOT_UNIQUE: A HED tag appears multiple times. A HED tag with *unique* attribute appears more than once in an event-level HED string.

HED_TAG_REPEATED: HED tags cannot be repeated in the same tag group or level. HED strings are not ordered, so *(B, C)* is equivalent to *(B, C)*.

(A, (A, B)) is not a duplicate.

(A, (B, C), A) and *(A, (B, C), (C, B))* are duplicates.

HED_TAG_REQUIRES_CHILD: A HED tag requires an additional ending node. The tag has the *requireChild* schema attribute but does not have a child.

HED_TILDES_UNSUPPORTED: The tilde notation is no longer supported. The **tilde syntax is no longer supported** for any version of HED. Annotators should replace the syntax *(A ~ B ~ C)* with *(A, (B, C))*.

The tilde (~) is considered an invalid character in all versions of the schema.

HED_UNITS_DEFAULT_USED: (WARNING) A HED tag value is missing units. If the corresponding unit class has default units, those are assumed.

HED_UNITS_INVALID: HED tag value has incorrect or invalid units. A HED tag has a value with units that are invalid or not of the correct unit class for the tag.

A typical mistake is to use unit modifiers with units that are not SI units.

HED_VALUE_INVALID: The value substituted for a placeholder (#) is not valid. A tag value is incompatible with the specified value class.

A tag value with no value class is assumed to be a label and may contain invalid characters.

A tag value is a schema node name.

HED_VERSION_DEPRECATED: (WARNING) The HED version is deprecated. It is strongly recommended that a current version be used as these deprecated versions may not be supported in the future.

Deprecated versions can be found in https://github.com/hed-standard/hed-specification/tree/master/hedxml/deprecated.

HED_VERSION_WARNING: (WARNING) The HED version number or HED schema was not provided or was invalid, so the latest version is used.

9.2 B.2. Schema validation errors

This section is organized by the type of schema format that results in the error. Errors that might be detected regardless of the schema format start with HED_SCHEMA. Errors that are specific to the `.mediawiki` format start with HED_WIKI. Errors that occur in the construction of the XML version or that are detected by XML validators when the planned XSD validation is implemented start with HED_XML.

9.2.1 B.2.2. General validation schema errors

HED_SCHEMA_ATTRIBUTE_INVALID: An attribute not defined in the appropriate schema section. The `unitClass` attribute must be defined in the `unitClassDefinitions` section of the schema.

A `unitClass` attribute has an invalid suffix because it is not a plural or unit modifier.

A `valueClass` attribute must be defined in the `valueClassDefinitions` section of the schema.

An schema attribute is not defined in the `schemaAttributeDefinitions` section.

HED_SCHEMA_CHARACTER_INVALID: The specification contains an invalid character.

HED_SCHEMA_DUPLICATE_NODE: A schema node name appears in the schema more than once.

HED_SCHEMA_HEADER_INVALID: The schema header is invalid. The head has invalid characters or format.

The header has unrecognized attributes.

HED_SCHEMA_NODE_NAME_INVALID: Schema node name is empty or contains invalid characters.

HED_SCHEMA_REQUIRED_SECTION_MISSING: A required schema section is missing. The required sections (corresponding to the schema, unit classes, unit modifiers, value classes, schema attributes, and properties) are not in the correct order.

Required schema sections may be empty, but still be given.

HED_SCHEMA_VERSION_INVALID: The schema version in the HED line or element is invalid. A HED version specification does not have the correct syntax for the schema file format.

A HED schema version does not comply with semantic versioning.

9.2.2 B.2.3. Format-specific schema errors.

HED_WIKI_DELIMITERS_INVALID: Delimiters used in the wiki are invalid. Schema line content after node name is not enclosed with <nowiki></nowiki> delimiters.

A line has unmatched or multiple <nowiki></nowiki>, `[ ]`, or `{ }` delimiters.

HED_WIKI_LINE_START_INVALID: Start of body line not `'''` or *.

HED_WIKI_SEPARATOR_INVALID: Required wiki section separator is missing or misplaced. A required schema separator is missing. (The required separators are: `!# start schema`, `!# end schema`, and `!# end hed`.)

HED_XML_SYNTAX_INVALID: XML syntax or does not comply with specified XSD.

9.2.3 B.3. Schema loading errors

Schema loading errors can occur because the file is inaccessible or is not proper XML. Schema loading errors are handled in different ways by the Python and JavaScript tools.

Python tools generally raise a `HedFileError` exception when a failure to load the schema occurs. The calling programs are responsible for deciding how to handle such a failure.

JavaScript tools in contrast are mainly used for validation in HED validation BIDS and are mainly called by the BIDS validator. Usually BIDS datasets provide a HED version number to designate the version of HED to be used, and the HED JavaScript validator is responsible for locating and loading schema.

BIDS validator users do not always have unrestricted access to the Internet during the validation process. The HED JavaScript tools have a fallback of the loading of the specified schema fails. The validator loads an internal copy of the most recent version of the HED schema and loads it. However, it also reports a **HED_SCHEMA_LOAD_FAILED** issue to alert the user that the schema used for validation may not be the one designated in the dataset. However, validation will continue with the fallback schema.

If the fallback schema stored with the HED validator fails to load, the **HED_SCHEMA_LOAD_FAILED** issue will also be reported and no additional HED validation will occur.

CHAPTER

TEN

DOCUMENTATION

10.1 1. HED publications

Explanation of the history, development, and motivation for third generation HED:

> Robbins, K., Truong, D., Jones, A., Callanan, I., & Makeig, S. (2021). Building FAIR functionality: Annotating events in time series data using Hierarchical Event Descriptors (HED). In press for Neuroinformatics Special Issue Building the NeuroCommons. https://doi.org/10.31219/osf.io/5fg73

Detailed case study in using HED-3G for tagging:

> Robbins, K., Truong, D., Appelhoff, S., Delorme, A., & Makeig, S. (2021, May 7). Capturing the nature of events and event context using Hierarchical Event Descriptors (HED). In press for NeuroImage Special Issue Practice in MEEG. https://authors.elsevier.com/sd/article/S1053-8119(21)01038-7.

10.2 2. Working documents

Mapping of HED terms and their descriptions to known ontologies is:

> HED-3G Working Document on Ontology mapping https://drive.google.com/file/d/13y17OwwNBlHdhB7hguSmOBdxn0Uk4hsI/view?usp=sharing

Two other working documents hold portions of the HED-3G specification that are under development and will not be finalized for Release 1:

> HED-3G Working Document on Spatial Annotation https://docs.google.com/document/d/1jpSASpWQwOKtan15iQeiYHVewvEeefcBUn1xipNH5-8/view?usp=sharing
>
> HED-3G Working Document on Task Annotation https://docs.google.com/document/d/1eGRI_gkYutmwmAl524ezwkX7VwikrLTQa9t8PocQMlU/view?usp=sharing

10.3 3. Schema viewers

The HED schema is usually developed in `.mediawiki` format and converted to XML for use by tools. However, researchers wishing to tag datasets will find both of these views hard to read. For this reason, we provide links to three versions of the schema. The expandable HTML viewer is easier to navigate. Annotators can also use CTAGGER, which includes a schema viewer and tagging hints.

Table 1: HED web-based schema vocabulary viewers.

Viewer	Link
Expandable HTML	https://www.hedtags.org/display_hed.html?version=8.0.0
Mediawiki	https://github.com/hed-standard/hed-specification/blob/master/hedwiki/HED8.0.0.mediawiki
XML	https://github.com/hed-standard/hed-specification/blob/master/hedxml/HED8.0.0.xml

10.4 4. HED Websites

The following is a summary of the HED-related websites

Table 2: HED websites.

Description	Site
Information and documentation	
HED organization website	https://www.hedtags.org
HED organization github	https://github.com/hed-standard
HED specification repository	https://github.com/hed-standard/hed-specification
Examples of HED annotation	https://github.com/hed-standard/hed-examples
HED documentation website	https://github.com/hed-standard/hed-standard.github.io
HED Python resources	
Python code repository	https://github.com/hed-standard/hed-python
Python validator and tools	https://github.com/hed-standard/hed-python/tree/master/hedtools
HED JavaScript resources	
HED JavaScript code	https://github.com/hed-standard/hed-javascript
BIDS validator	https://github.com/bids-standard/bids-validator
HED Matlab resources	
Matlab source code	https://github.com/hed-standard/hed-matlab
Annotator resources	
CTAGGER executable jar	https://github.com/hed-standard/hed-java/raw/master/ctagger.jar
CTAGGER repository	https://github.com/hed-standard/CTagger
Java repository	https://github.com/hed-standard/hed-java
Online HED tools	
Online website	https://hedtools.ucsd.edu/hed
Docker deployment	https://github.com/hed-standard/hed-python/tree/master/webtools/deploy_hed

CHAPTER

ELEVEN

TOOLS AND SERVICES

11.1 1. CTagger for annotation

The CTagger tool for annotating data provides a graphical user interface (GUI) to assist HED users in the annotation process. CTagger can be run as a standalone application (https://github.com/hed-standard/hed-java/raw/master/ctagger.jar) or using the HEDTools plug-in in EEGLAB.

The tool is designed to ease the process of constructing HED strings, with features including tag search, an expandable schema-browser view, and free-form formatting. The interchangeability between long-short forms introduced in HED-3G is fully supported. CTagger is also compatible with BIDS, allowing users to import BIDS `events.tsv` and `events.json` files to extract the event structure. Once finished tagging, users can export their HED annotation into a json file compatible with BIDS events.json. See CTagger GitHub repository for more details, guides, and tutorials.

11.2 2. HED online tools

The web-based tools are summarized in this section. All tools are available from the main access point https://hedtools.ucsd.edu/hed. The services are implemented in a Docker module and can be deployed locally provided that Docker is installed.

Event files are BIDS style tab-separated value files. The first line is always a header line giving the names of the columns, which are used as keys to metadata in accompanying JSON sidecars. The online tools for BIDS events file are designed to help users debug their HED annotations for BIDS datasets before using the BIDS validator.

Web tools for BIDS-style events.tsv files

Assemble events: Assemble HED annotation of a BIDS-style events file.

- Upload the event file and an optional JSON sidecar file.
- Specify the HED version.
- Select the `Assemble` processing option and whether to expand definitions.
- Click the `Process` button.
- The tool assembles a new two-column event file.
- If there are errors, a downloadable file of error messages is returned.
- If no errors, the new event file is returned.

Validate events: Validate a BIDS-style events file with optional JSON sidecar.

- Upload the event file and an optional JSON sidecar file.

- Specify the HED version.
- Select the `Validate` processing option.
- Click the `Process` button.
- The tool first validates the sidecar if present.
- If the sidecar contains no errors, the tool validates the events file with the sidecar.
- If there are any errors, the tool returns a downloadable file of error messages.

Notes:

1. If the HED version number is given, the tool downloads a standard version from GitHub. Otherwise, the user must upload a local HED schema.
2. The `Assemble` creates a two-column event file. The first column is the event onset time and the second column is the full event-level HED annotation.

HED schema tools are designed to assist HED schema developers and library schema developers in making sure that the schema has the correct form and to provide easy conversion between schema formats.

Web tools for HED schema files.

Convert schema: Convert a HED schema between XML and MEDIAWIKI format.

- Upload a HED schema file or gives a URL pointing to a schema file.
- Select the `Convert` option.
- Press the `Process` button.
- The tool returns a downloadable converted file (XML input is converted to MEDIAWIKI and vice versa).
- Errors are reported as message at the bottom of the screen.

Validate schema: Validate a HED schema between XML and MEDIAWIKI format.

- Upload a HED schema file or gives a URL pointing to a schema file.
- Selects the `Validate` option.
- Press the `Process` button.
- The tool returns a downloadable file of error messages if the schema is invalid.

BIDS JSON sidecars have file names ending in `_events.json`. These JSON files contain metadata and HED tags applicable to associated events files.

Web tools for BIDS style JSON sidecar.

Convert to long: Convert the HED tags in a BIDS-style events JSON sidecar to long form.

- Upload the JSON sidecar file.
- Specify the HED version of HED.
- Select the `Convert to long` option.
- Press the `Process` button.

- The tool first validates the sidecar and returns an error file if errors.
- Otherwise, the tool returns a JSON sidecar with the HED tags converted to full-paths.

Convert to short: Convert the HED tags in a BIDS-style events JSON sidecar to short form.

- Upload the JSON sidecar file.
- Specify the HED version.
- Select the `Convert to short` option.
- Press the `Process` button.
- The tool first validates the sidecar and returns an error file if errors.
- Otherwise, the tool returns a JSON sidecar with the HED tags converted to short-form.

Validate sidecar: Validate a single BIDS-style events JSON sidecar.

- Upload the JSON sidecar file.
- Specify the HED version.
- Select the `Validate` option.
- Press the `Process` button.
- The tool validates the sidecar and returns an error file if there are errors.

Notes:

1. If the HED version number is given, the tool downloads a standard version from GitHub. Otherwise, the user must upload a local HED schema.

Spreadsheets (either in Excel or tab-separated-value format) are convenient for organizing tags.

Web tools for spreadsheets of HED tags.

Convert to long: Convert the HED tags in tag spreadsheet to long form.

- Upload the spreadsheet file, indicating whether the first row is a header.
- Select a worksheet if the spreadsheet is an Excel file with multiple worksheets.
- Specify the HED version.
- Select the columns containing HED tags or are prefix columns.
- Select the `Convert to long` option.
- Press the `Process` button.
- The tool first validates the spreadsheet and returns an error file if errors.
- Otherwise, the tool returns a spreadsheet with the HED tags converted to full-paths.

Convert to short: Convert the HED tags in a spreadsheet to short form.

- Upload the spreadsheet file, indicating whether the first row is a header.
- Select a worksheet if the spreadsheet is an Excel file with multiple worksheets.
- Specify the HED version.
- Select the columns containing HED tags or are prefix columns.

- Select the `Convert to short` option._ Press the `Process` button.
- The tool first validates the spreadsheet and returns an error file if errors.
- Otherwise, the tool returns a spreadsheet with the HED tags converted to short-form.

Validate: Validate the HED tags in a spreadsheet.

- Upload the spreadsheet file, indicating whether the first row is a header.
- Selects a worksheet if the spreadsheet is an Excel file with multiple worksheets.
- Specify the HED version.
- Select the columns containing HED tags or are prefix columns.
- Select the `Validate` option.
- Press the `Process` button.
- The tool validates the spreadsheet and returns an error file if the spreadsheet is invalid.

Notes:

1. If the HED version number is given, the tool downloads a standard version from GitHub. Otherwise, the user must upload a local HED schema.

Online validation of HED strings.

Web tools for processing strings

Convert to long: Convert a HED string to long form.

- Paste a string into the input text box.
- Specify the HED version.
- Select the `Convert to long` option.
- Press the `Process` button.
- The tool first validates the string and returns errors in the lower text box if any.
- Otherwise the tool returns the full path version of the strings in the lower text box.

Convert to short: Convert a HED tag string to short form.

- Paste a string into the input text box.
- Specify the HED version.
- Selects the `Convert to short` option.
- Press the `Process` button.
- The tool first validates the string and returns errors in the lower text box if any.
- Otherwise the tool returns the short-form versions of the tag strings in the lower text box.

Validate: Validate a HED string.

- Paste a string into the input text box.
- Specify the HED version.
- Select the `Validate` option and presses the `Process` button.

- The tool validates the string and returns any error messages in the lower text box.

Notes:

1. If the HED version number is given, the tool downloads a standard version from GitHub. Otherwise, the user must upload a local HED schema.

11.3 3. HED REST services

HED supports a number of RESTful web services in support of HED including schema conversion and validation, JSON sidecar validation, spreadsheet validation, and validation of a single BIDS event file with supporting JSON sidecar. Short-to-long and long-to-short conversion of HED tags are supported for HED strings, JSON sidecars, BIDS-style events files and spreadsheets in `.tsv` format. Support is also included for assembling the annotations for a BIDS-style event file with a JSON sidecar. There is also support for extracting a template of a JSON sidecar from a BIDS events file.

The `hedexamples/matlab` directory of the `hed-python` repository gives running MATLAB examples of how to call these services in MATLAB.

11.3.1 3.1 Service setup

The HED web services are accessed by making a request to the HED web server to obtain a CSRF access token for the session and then making subsequent requests as designed. The steps are:

1. Send an HTTP `get` request to the HED CSRF token access URL.
2. Extract the CSRF token and returned cookie from the response to use in the headers of future `post` requests.
3. Send an HTTP `post` request in the format as described below and read the response.

The following table summarizes the location of the relevant URLs for online deployments of HED web-based tools and services.

Table 1: URLs for HED online services.

Service	URL
Online HED tools	https://hedtools.ucsd.edu/hed
CSRF token access	https://hedtools.ucsd.edu/hed/services
Service request	https://hedtools.ucsd.edu/hed/services_submit

11.3.2 3.2 Request format

HED services are accessed by passing a JSON dictionary of parameters in a request to the online server. All requests are in JSON format and include a `service` name and additional parameters.

The service names are of the form `target_command` where `target` specifies the input data type and `command`specifies the service to be performed. For example, `events_validate` indicates that a BIDS-style event file is to be validated. The exception to this naming rule is`get_services`.

All parameter values are passed as strings. The contents of file parameters are read into strings to be passed as part of the request. The following example shows the JSON for a HED service request to validate a JSON sidecar. The contents of the JSON file to be validated are abbreviated as `"json file text"`.

Example: Request parameters for validating a JSON sidecar.

```
{
    "service": "sidecar_validate",
    "schema_version": "8.0.0",
    "json_string": "json file text",
    "check_for_warnings": "on"
}
```

The parameters are explained in the following table. Parameter values listed in square brackets (e,g, [a, b]) indicate that only one of a or bshould be provided.

Table 2: Summary of HED ReST services

Service	Parameters	Descriptions
get_services	none	Returns a list of available services.
events_assemble	events_string,json_string,[schema_string,schema_version],check_for_warnings,defs_expand	Assemble tags from machine-actionable BIDS events file and single HED string.Returned data: a file of assembled events as text or an error file as text if errors.
events_extract	events_string	Extract a template JSON sidecar based on the contents of the event file.Returned data: A JSON sidecar (template) if no errors..
events_validate	events_string,json_string,[schema_string,schema_version],check_for_warnings	Validate a BIDS-style events file and its JSON sidecar if provided.Returned data: an error file as text if errors.
sidecar_to_long	json_string,[schema_string,schema_version]	Convert a JSON sidecar with all of its HED tags expressed in long form.Returned data: a converted JSON sidecar as text or an error file as text if errors.
sidecar_to_short	json_string,[schema_string,schema_version]	Convert a JSON sidecar with all of its HED tags expressed in short form.Returned data: a converted JSON sidecar as text or an error file as text if errors.
sidecar_validate	json_string,[schema_string,schema_version],check_for_warnings	Validate a BIDS-style JSON sidecar.Returned data: an error file as text if errors.
spreadsheet_to_long	spreadsheet_string,[schema_string,schema_version],check_for_warnings,column_x_check,column_x_inpu	Convert a tag spreadsheet (tsv only) to long form.Returned data: converted tag spreadsheet as text if no errors.
spreadsheet_to_short	spreadsheet_string,[schema_string,schema_version],check_for_warnings,column_x_check,column_x_inpu	Convert a tag spreadsheet (tsv only) to short form.Returned data: converted tag spreadsheet as text if no errors.
spreadsheet_validate	spreadsheet_string,[schema_string,schema_version],check_for_warnings,column_x_check,column_x_inpu	Validate a tag spreadsheet (tab-separated format only).Returned data: an error file as text if errors.
strings_to_long	string_list,[schema_string,schema_version]	Return list of strings to long form.
strings_to_short	string_list,[schema_string,schema_version]	Convert list of long-form strings to short-form strings.
strings_validate	hed_strings,[schema_string,schema_version]	Validate a list of HED strings and returns a list of errors.

The following table gives an explanation of the parameters used for various services.

Table 3: Parameters for web services.

Key value	Type	Description
check_for_warnings	boolean	If true, check for warnings when processing.
column_x_check:	boolean	If present with value 'on', column x has HED tags.".
column_x_input:	string	Contains the prefix prepended to column x if column x has HED tags.
defs_expand	boolean	If true replaces *def/XXX* with *def-expand/XXX* grouped with the definition content.
events_string	string	A BIDS events file as a string..
hed_columns	list of numbers	A list of HED string column numbers (starting with 1).
hed_schema_string	string	HED schema in XML format as a string.
hed_strings	list of strings	A list containing HED strings.
json_string	string	BIDS-style JSON events sidecar as a string.
json_strings	string	A list of BIDS-style JSON sidecars as strings.
schema_string	string	A HED schema file as a string.
schema_version	string	Version of HED to be accessed if relevant.
service	string	The name of the requested service.
spreadsheet_string	string	A spreadsheet tsv as a string.

11.3.3 3.3 Service responses

The web-services always return a JSON dictionary with four keys: `service`, `results`, `error_type`, and `error_msg`. If `error_type` and `error_msg` are not empty, the operation failed, while if these fields are empty, the operation completed. Completed operations always return their results in the `results` dictionary. Keys in the `results` dictionary return as part of a HED web service response.

Table 4: The results dictionary.

Key	Type	Description
command	string	Command executed in response to the service request.
command_target	string	The type of data on which the command was executed..
data	string	A list of errors or the processed result.
schema_version	string	The version of the HED schema used in the processing.
msg_category	string	One of success, warning, or failure depending on the result.
msg	string	Explanation of the result of service processing.

The `msg` and `msg_category` pertain to contents of the response information. For example a `msg_category` of `warning` in response to a validation request indicates that the validation completed and that the object that was validated had validation errors. In contrast, the `error_type`, and `error_msg` values are only for web service requests. These keys indicate whether validation was able to take place. Examples of failures that would cause errors include the service timing out or the service request parameters were incorrect.

11.4 4. Python tools

The python code for validation is in the hedtools project located in the hed-python repository https://github.com/hed-standard/hed-python. You can install the tools using pip if you have downloaded the hed-python repository:

```
pip install <hedtools-local-path>
```

The validation functions are in the hed.validator module. The data representations for various items such as dictionaries or event files can be found in the hed.models module. The hed_input.py module reads in a spreadsheet and possibly a dictionary and creates a HedInput object representing the spreadsheet. The hed-validator.py module creates a HedValidator object that takes a HedSchema object to use in subsequent validation. The validate_input method of HedValidator validates HED input in various formats and returns a list of issues.

11.5 5. JavaScript tools

The JavaScript code for HED validation is in the validation directory of the hed-javascript repository located at https://github.com/hed-standard/hed-javascript.

11.5.1 5.1 Installation

You can install the validator using npm:

```
npm install hed-validator
```

11.5.2 5.2 Package organization

This package contains two sub-packages.

hedValidator.validator validates HED strings and contains the functions:

> buildSchema imports a HED schema and returns a JavaScript Promise object.validateHedString validates a single HED string using the returned schema object.

hedValidator.converter converts HED strings between short and long forms and contains the following functions:

> buildSchema behaves similarly to the buildSchema function in hedValidator.validator except that it does not work with attributes.
>
> convertHedStringToShort converts HED strings from long form to short form.
>
> convertHedStringToLong converts HED strings from short form to long form.

11.5.3 5.3 Programmatic interface

The programmatic interface to the HED JavaScript buildSchema must be modified to accommodate a base HED schema and arbitrary library schemas. The BIDS validator will require additional changes to locate the relevant HED schemas from the specification given by "HEDVersion" in dataset_description.json.

The programmatic interface is similar to the JSON specification of the proposed BIDS implementation except that the "fileName" key has been replaced by a "path" key to emphasize that callers must replace filenames with full paths before calling buildSchema.

Example: JSON passed to buildSchema.

```
{
    "path": "/data/wonderful/code/mylocal.xml",
    "libraries": {
        "la": {
            "libraryName": "libraryA",
            "version": "1.0.2"
        },
        "lb": {
            "libraryName": "libraryB",
            "path": "/data/wonderful/code/HED_libraryB_0.5.3.xml"
        }
    }
}
```

NOTE: This interface is proposed and is awaiting resolution of BIDS PR #820 on file passing to BIDS.

11.6 6. MATLAB tools

HED validation can be done using the online web-services from MATLAB as shown in the `./examples/matlab` directory of the hedweb project in the hed-python repository.