HED specification

Release 3.2.0

HED Working Group

Apr 26, 2024

CONTENTS:

Links

- PDF released version (V3.2.0)
- PDF working version
- Specification source
- Stable specification source
- HED resources
- HED homepage
- HED project homepage

SPECIFICATION ROLE

The HED specification document formalizes the syntax and behavior of HED (Hierarchical Event Descriptors) vocabulary, annotations, and supporting tools. The specification supports three versions of the specification:

- develop - development branch which is under discussion.

- latest - includes revisions approved by the HED Working Group but not released.

- stable - the latest released form.

For more information about HED see The HED project homepage and the HED resources page.

1.1 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 specification applies to HED Schema versions > 8.0.0 and above.

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** forum on the **hed-specification** GitHub repository to report issues with this specification document.

For requests for additional features and vocabulary enhancements of the HED schema use the **issues** forum on the **hed-schemas** GitHub repository.

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 1.1. 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 interoperable 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 and tools*.

1.1.2 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 < 4.0.0) 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 4.0.0 - 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. The HED-2G represents a **sub-tag** system.

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. Use 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, *HED library schemas* were introduced to allow detailed annotation of terms importance to individual user communities without complicating the standard schema. 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 have specialized vocabulary requirements. 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.

1.1.3 1.2. Goals of HED

An event is a process that unfolds over time and represents something that happens. Events are typically measured by noting sequences of time points (event markers) marking specific transition points.

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** such as noise and vibration from construction work unrelated to the experiment, 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.

1.1.4 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.

1.1.5 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: HED formats* specifies the formats for HED vocabularies and annotations. Basic and advanced event models and their annotations are explained in *Chapter 4: Basic annotation* and *Chapter 5: Advanced annotation*. *Chapter 6: Infrastructure and tools* discussions how tags should be handled by HED-compliant tools. *Chapter 7: Library schemas* discusses the basic rules for library schema creation.

Appendix A: Schema format provides a reference manual for the HED schema format rules, and *Appendix B: HED errors* gives a complete listing of HED error codes and their meanings. A common set of test cases for these errors is available in the **tests** directory of the **hed-specification** GitHub repository.

Other resources include a comprehensive list of **HED resources** including additional documentation, tutorials and code examples.

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

1.2 2. HED terminology

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this specification 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.

1.2.1 2.1 Definitions

1.2.1.1 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).

1.2.1.2 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.

1.2.1.3 Control-variable [*]

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

1.2.1.4 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.

1.2.1.5 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).

1.2.1.6 Event-context [*]

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

1.2.1.7 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.

1.2.1.8 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.

1.2.1.9 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.

1.2.1.10 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.

1.2.1.11 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.)

1.2.1.12 HED string

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

1.2.1.13 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.

1.2.1.14 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.

1.2.1.15 Parameter [*]

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

1.2.1.16 Recording [*]

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

1.2.1.17 Tag-group

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

1.2.1.18 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.

1.2.1.19 Temporal scope

The time interval between the start and end of an event process. Often the start time is annotated with the `Onset` tag, and the end time is annotated with the `Offset` tag. Since in practical terms the time is measured in discrete samples, the temporal scope includes the start time sample but does not include the end time sample.

1.2.1.20 Time-block [*]

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

1.2.2 2.2 Character sets and restrictions

Starting with HED standard schema versions 8.3.0 and above, HED will allow UTF-8 characters in various settings. The types of characters referred to in this specification are:

Name	Description
`ascii`	utf-8 codes 0 to 127 (single byte)
`nonascii`	utf-8 codes greater than 128 (multi-byte)
`printable`	ASCII 32 <= code < 127
`lowercase`	ASCII characters a-z
`uppercase`	ASCII characters A-Z

Table 1 – continued from previous page

Name	Description
letters	lowercase and/or uppercase
text	printable and/or nonascii excluding comma, square brackets, and curly braces.
digits	0-9
tab	ASCII code 09
newline	ASCII code 10 (linefeed)
blank	ASCII code 32
exclamation	ASCII code 33
double-quote	ASCII code 34
number-sign	ASCII code 35
dollar	ASCII code 36
percent-sign	ASCII code 37
ampersand	ASCII code 38
single-quote	ASCII code 39
left-paren	ASCII code 40
right-paren	ASCII code 41
asterisk	ASCII code 42
plus	ASCII code 43
comma	ASCII code 44
hyphen	ASCII code 45
period	ASCII code 46
slash	ASCII code 47
colon	ASCII code 58
semicolon	ASCII code 59
less-than	ASCII code 60
equals	ASCII code 61
greater-than	ASCII code 62
question-mark	ASCII code 63
at-sign	ASCII code 64
backslash	ASCII code 92
caret	ASCII code 94
underscore	ASCII code 95
verical-bar	ASCII code 124
tilde	ASCII code 126
alphanumeric	letters and/or digits
name	alphanumeric, hyphen, period, underscore, nonascii

1.3 3. HED formats

This chapter describes the requirements and formats for HED schema and HED annotations.

1.3.1 3.1. Schema formats

A **HED schema** is a formal specification of a HED vocabulary and annotation format rules. A HED schema vocabulary is organized hierarchically so that similar concepts and terms appear close to one another in the organizational hierarchy.

HED schema nodes **must** satisfy an "is-a" relationship with their parent nodes in the schema. That is, if node *A* is an ancestor of node *B* in the schema, then *B* is a type of *A*. This relationship is fundamental to HED and permits search generality. Searches for *A* are able to also return instances of *B*.

A key requirement for third generation HED (versions >=8.0.0) is that all node names (tag terms) in the HED schema (except for # placeholders) **must be unique**.

Additional details about HED schema format can be found in appendix *A. Schema format details*. *7. Library schemas* discusses the additional requirements and restrictions on library schemas.

B.2.   Schema validation errors gives the errors Library specific schema issues usually generate *SCHEMA_LIBRARY_INVALID* errors.

1.3.1.1 3.1.1. Official schema releases

The HED ecosystem supports a standard base schema and additional discipline-specific library schemas. (See the **expandable schema viewer** to explore existing schemas.)

Releases of the HED standard base schema are stored in **standard_schema/hedxml** directory of the **hed-schemas** repository.

Releases of a HED library schemas are stored in a subdirectory of **library_schemas** whose name is the library name.

1.3.1.2 3.1.2. Schema layout overview

Schemas can be specified in either `.mediawiki` or `.xml` format. The HED schema **online tools** provide an easy way for users to validate schema and convert between formats.

HED schema developers usually use `.mediawiki` format for more convenient editing, display, and viewing on GitHub. However, the stable links provided for tools to access and download the HED schema are to the XML versions. Both formats must be available and synchronized in the **hed/standard/hed-schemas** GitHub repository.

Regardless of the format, a valid HED schema must have the following sections in this order:

Required sections of a HED schema (in the required order):

Section	Mediawiki format	XML format
Header line	`HED version="8.0.0"`	`<HED version="8.0.0">`
Prologue	`'''Prologue'''`	`<prologue> ... </prologue>`
Schema start	`!# start schema`	`<schema>...`
Schema end	`!# end schema`	`</schema>`
Unit classes	`'''Unit classes'''`	`<unitClassDefinitions>... </unitClassDefinitions>`
Unit modifiers	`'''Unit modifiers'''`	`<unitModifierDefinitions>    ...    </unitModifierDefinitions>`
Value classes	`'''Value classes'''`	`<valueClassDefinitions>... <valueClassDefinitions>`
Schema attributes	`'''Schema attributes'''`	`<schemaAttributeDefinitions>    ...    </schemaAttributeDefinitions>`
Properties	`'''Properties'''`	`<propertyDefinitions>... </propertyDefinitions>`
Epilogue	`'''Epilogue'''`	`<epilogue>... </epilogue>`
Ending line	`!# end hed`	`</HED>`

The sections in the `.xml` version must always be terminated by closing `</ >` tokens, whereas the sections of the `.mediawiki` version, which is line-oriented, are terminated when the next section begins (`#!`) or a top tag (`'''`) is encountered.

The actual HED tag specifications (referred to in the discussion as nodes or tag terms) appear in the `schema` section, while the remaining sections specify additional information and behavior. These additional sections are required, but are allowed to be empty.

If any of the required sections of the schema are missing or out of order, a *SCHEMA_SECTION_MISSING* error occurs.

Each of the schema sections has "schema attributes", which are the attributes that may be assigned to elements in a given section. If a schema attribute is applied improperly to an element in a given section, the *SCHEMA_ATTRIBUTE_INVALID* error occurs.

See *Appendix A. Schema format details* for additional details.

3.1.2.1. The header

The schema header line MUST specify the `version` attribute whose value MUST be a valid semantic version. See *SCHEMA_VERSION_INVALID*.

A schema may optionally contain `library`, `withStandard`, and `unmerged` attributes for library schemas. A schema's library name or lack there of is used to locate the schema in the **hed-schemas** GitHub repository.

The header may optionally contain an XSD namespace specification. If the schema contains any additional unrecognized attributes, *SCHEMA_HEADER_INVALID* error occurs.

See *A.2.2. MediaWiki header* and *A.3.2. XML header* for more detailed information on the MediaWiki and XML header formats, respectively.

3.1.2.2. The prologue

The prologue should contain a concise introduction to the schema and its purpose. Together with *the epilogue* section, the contents are used by tools to provide information about the schema to the users.

The prologue may contain `text` characters or `newline`. If other characters appear, a *SCHEMA_CHARACTER_INVALID* error occurs.

3.1.2.3. The schema section

The schema section contains the actual vocabulary contents of the schema. Each element in this section is a *node* element, which we will also call a *tag term*. The location of the node element within the section specifies its relationship to other tag terms in the schema.

A node element specifies a name, node attributes, and an informative description of the tag term's meaning. A node name may only contain valid `name` characters (`alphanumeric`, `hyphen`, `underscore`, `period`, and `nonascii`).

This also applies to tag extensions. Substitutions for the `#` placeholder that have value classes are governed by the rules of that value class. If other characters appear, a *SCHEMA_CHARACTER_INVALID* error occurs.

Each schema node element must be unique or a *SCHEMA_DUPLICATE_NODE* error is generated.

3.1.2.4. Unit classes and units

The unit classes are attributes that modify the # schema placeholder nodes. The unit class definition section specifies the allowed unit classes for the schema as well as the associated units that can be used with tags that take values.

Only the singular version of each unit is explicitly specified, but the corresponding plurals of the explicitly mentioned singular version 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.

Units may be in one of four forms as designated by their unit type attributes:

Unit type	Unit type attributes
SI unit	only `SIUnit`
SI unit symbol	both `SIUnit` and `unitSymbol`
unit that is not an SI unit	no unit type attribute
unit symbol is not an SI unit	only `unitSymbol`

Most units appear after the value in annotations. However, certain units such as $ appear before their corresponding values. These units have the `unitPrefix` attribute.

If a unit class, `SIUnit`, or `unitPrefix` attribute appears in a section other than the unit class definition section of the schema, a *SCHEMA_ATTRIBUTE_INVALID* error occurs. See appendix *A.1.1. Unit classes and units* for additional details and a listing.

Units names are case-insensitive and should not contain blanks. Unit symbols MUST maintain their case. Unit class names are case-insensitive, but MUST contain only valid `name` characters. If other characters appear, a *SCHEMA_CHARACTER_INVALID* error occurs.

3.1.2.5. Unit modifiers

The unit modifier definition section lists the SI unit multiples and submultiples that are allowed to be prepended to units that have the `SIUnit` schema attribute.

Unit modifiers can only be used with SI units and SI unit symbols. SI unit modifiers used with ordinary SI units have the `SIUnitModifier` attribute, while unit modifiers used with SI unit symbols have the `SIUnitSymbolModifier` attribute.

If a `SIUnitModifier`, or `SIUNitSymbolModifier` attribute appears in a section other than unit modifier section of the schema, a *SCHEMA_ATTRIBUTE_INVALID* error occurs.

Unit modifiers are case-sensitive.

See appendix *A.1.2. Unit modifiers* for additional details and a listing of values for the standard schema.

3.1.2.6. Value classes

The value class definition 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 take `textClass` values.

See appendix *A.1.3. Value classes* for additional details and a listing of values for the standard schema.

Value class names are insensitive, but must contain only valid `name` characters.

3.1.2.7. Schema attributes

The schema attribute definition section lists the schema attributes that may be applied to schema elements in other sections of the schema (except for the properties section).

The specification of which type of schema elements a particular schema attribute may apply to is specified by its schema properties. If a schema attribute appears in a section contradicted by its properties, a *SCHEMA_ATTRIBUTE_INVALID* error occurs.

See appendices *A.1.4. Schema attributes* and *A.1.5. Schema properties* for additional details and a listing for the standard schema.

3.1.2.8. Schema properties

The schema properties section lists the allowed properties of the schema attributes. These properties help tools validate certain requirements directly based on the HED schema rather than on a hard-coded implementation.

There are two types of properties: **form type** and **section type** properties. The `boolProperty` is a form type property indicating that a schema attribute does not take a value. Rather, its presence indicates true and absence indicate false.

The *section* type properties indicate the sections in which a schema attribute may appear. The section properties include `unitClassProperty`, `unitModifierProperty`, `unitProperty`, and `valueClassProperty`. Schema attributes without any section properties are assumed to apply to node elements.

A schema attribute may have multiple section properties, indicating that the attribute may appear as an attribute in multiple sections of the schema.

See *A.1.4 Schema attributes* and *A.1.5. Schema properties* for information and a listing of schema attributes and their respective properties.

3.1.2.9. The epilogue

The epilogue should give license information, acknowledgments, and references.

The epilogue may contain `text` characters or `newline`. If other characters appear, a *SCHEMA_CHARACTER_INVALID* error occurs.

1.3.1.3 3.1.3. Naming conventions

The different parts of the HED schema have different rules for the characters and the names that are allowed.

3.1.3.1. Node elements

Schema designers and users that extend HED schema or develop library schema will be mainly concerned with nodes (tag terms) found in the schema section. The names of these elements must conform to the rules for `nameClass`.

Other conventions and requirements for the contents of schema node elements are as follows:

Naming conventions for nodes (tag terms) in HED schema.

1. By convention, the first letter of a schema node (tag term) should be capitalized with the remainder lower case.

2. Schema node names consisting of multiple words should not not contain blanks and should be hyphenated.

3. Schema descriptions should be concise sentences, possibly with clarifying examples.

---

4. Schema descriptions may include only `text` characters but should not contain `newline` or square brackets or braces.

3.1.3.2. Epilogue and prologue

The epilogue and prologue section text must conform to the rules for `text` value. The section text may have new lines, which are preserved.

3.1.3.3. Naming in other blocks

The names of elements corresponding to schema attributes, schema properties, unit classes, and value classes should start with a lower case letter, with the remainder in camel case.

Units and unit modifiers follow the naming conventions of the units they represent.

Case is preserved for unit modifiers, as uppercase and lowercase versions often have distinct meanings. The case for unit symbols is also maintained.

1.3.1.4 3.1.4. Mediawiki schema format

Mediawiki is a markdown-like format that was selected as the HED schema editing format because of its flexibility and ability to represent nested or hierarchical relationships.

The format is line-oriented, so each schema entry should be on a single line.

The schema must follow the layout described in the previous section. All sections are required, although they may be empty.

Top nodes in the schema are enclosed by pairs of three single quotes (`'''`). The levels of other nodes are designated by the number of asterisks (*) at the beginning of the respective defining lines. Each term is separated from its level-indicating asterisks by a single space.

Descriptions, which are enclosed in square brackets (`[ ]`), indicate the meaning of the item they modify. The descriptions are displayed to users by schema browsers and other tools, so every effort should be made to make them informative and clear.

Attributes are enclosed with curly braces (`{ }`). These attributes provide additional rules about how the item and modifying values should be used and handled by tools.

If an attribute or property is referenced in the schema, it must be defined in the appropriate definition section of the schema, or schema processing tools will generate a *SCHEMA_ATTRIBUTE_INVALID* error.

Allowed HED node attributes include unit class and value class values as well as HED schema attributes that do not have one of the following modifiers: `unitClassProperty`, `unitModifierProperty`, `unitProperty`, or `valueClassProperty`. Note: schema attributes having the `elementProperty` may apply anywhere in the schema, including the schema header, schema attributes having the `nodeProperty` may only apply to node elements.

HED schema attributes that have the `boolProperty` appear with just their name in the schema element they are modifying. The presence of such an attribute indicates that it is true or present.

HED schema attributes that do not have the `boolProperty` are specified in the form of a `name=value` pair. If multiple values of a particular attribute are applicable, they should be specified as name-value pairs separated by commas within the curly braces.

The following example shows a simple HED schema in `.mediawiki` format.

Example: Example HED schema in .mediawiki format.

```
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>{suggestedTag=Task-event-role,suggestedTag=Sensory-presentation}
→[Something perceivable by an agent.]</nowiki>
                              . . .
'''Property'''<nowiki>{extensionAllowed}[A characteristic.] </nowiki>
* Informational-property <nowiki>[A quality pertaining to information.]</nowiki>
** Label <nowiki>[A string of 20 or fewer characters.]</nowiki>
*** <nowiki># {takesValue}</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>
* extensionAllowed <nowiki>{boolProperty}[Attribute indicating that users can add child
→nodes.]</nowiki>
* suggestedTag <nowiki>[Attribute indicating another tag that is often associated with
→this tag.]</nowiki>
* takesValue <nowiki>{boolProperty}[Attribute indicating a placeholder to be replaced by
→a user-defined value.] </nowiki>
                  . . .
'''Properties''' <nowiki>[Properties of the schema attributes.]</nowiki>
* boolProperty <nowiki>[Indicates a schema attribute represents a boolean.]</nowiki>
                  . . .
'''Epilogue'''
An optional section that is the place for notes and is ignored in HED processing.

!# end hed
```

In the above example, `Property` in the `schema` section is a top node because it appears enclosed by three single quotes, while `Informational-property` is a first-level node because its defining line begins with a single asterisk (*).

`Sensory-event` in the `schema` section has a `suggestedTag` attribute (shown in curly braces). Similarly, `Property` has an `extensionAllowed` attribute, and the # placeholder has a `takesValue` attribute. The `schema attributes` section must include definitions of `suggestedTag`, `extensionAllowed` and `takesValue` or the schema will not validate.

The definition of the `takesValue` attribute has `boolProperty`, so a definition of `boolProperty` must be included in the `Properties` section or the schema will not validate.

Everything after each HED node (tag term) must be enclosed by <nowiki></nowiki> markup elements. The contents within these markup elements include the description and attributes.

Within the HED schema a # node indicates that the user must supply a value consistent with the unit and value class attributes of the # node during annotation. Lines with hashtag (#) placeholders should have everything after the asterisks, including the # placeholder, enclosed by <nowiki></nowiki> markup elements.

Additional details and rules can be found in appendix *A.2 Mediawiki file format*

1.3.1.5 3.1.5. XML schema format

The .xml format directly mirrors the order and information in the .mediawiki version of the schema.

The <node> elements of the schema represent the HED tags (tag terms), with remaining schema elements specifying additional information and properties.

Each <node> element must have a <name> child element corresponding to the HED tag term that it specifies.

A <node> element should also have a <description> child element whose content corresponds to 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. These <attribute> elements always have a <name> element child and also have a <value> element if the corresponding schema attribute does not have boolProperty.

The following is a translation of the .mediawiki example from the previous section in the HEDXML format.

Example: XML version of the example schema in the previous section.

```xml
<?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>
            <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>
```

(continues on next page)

```
                            <description>A string of less than 20.</description>
                            <node>
                                <name>#</name>
                                <attribute>
                                    <name>takesValue</name>
                                </attribute>
                            </node>
                        </node>
                </node>
            </node>
        </schema>
        <unitClassDefinitions></unitClassDefinitions>
        <unitModifierDefinitions></unitModifierDefinitions>
        <valueClassDefinitions></valueClassDefinitions>
        <schemaAttributeDefinitions>
            <schemaAttributeDefinition>
                <name>extensionAllowed</name>
                <description>Attribute indicating that users can add child nodes.</
→description>
                <property>
                    <name>boolProperty</name>
                </property>
            </schemaAttributeDefinition>
            <schemaAttributeDefinition>
                <name>suggestedTag</name>
                <description>Attribute indicating another tag that is often associated with␣
→this tag.</description>
            </schemaAttributeDefinition>
            <schemaAttributeDefinition>
                <name>takesValue</name>
                <description>Attribute indicating a placeholder to be replaced by a user-
→defined value.</description>
                <property>
                    <name>boolProperty</name>
                </property>
            </schemaAttributeDefinition>
        </schemaAttributeDefinitions>
        <propertyDefinitions>
            <propertyDefinition>
                <name>boolProperty</name>
                <description>Attribute indicating a placeholder to be replaced by a user-
→defined value.</description>
            </propertyDefinition>
        </propertyDefinitions>
        <epilogue>This epilogue is a place for notes and is ignored in HED processing.</
→epilogue>
</HED>
```

Additional details and rules can be found in appendix *A.3 XML file format*

1.3.2 3.2. Annotation formats

HED annotations are comma-separated strings of HED tags drawn from a HED schema vocabulary. 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.

1.3.2.1 3.2.1. Vocabulary organization

HED (Hierarchical Event Descriptors) are nodes (tag terms) organized hierarchically under their respective root or **top nodes**. In HED versions >= 8.0.0 these top nodes are: `Event`, `Agent`, `Action`, `Item`, `Property`, and `Relation`. Each top node and its subtree represent distinct **is-a** relationships for the vocabulary schema.

The `Event` subtree tags indicate the general event category, 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 and the annotations associated with a single event marker may represent multiple events.

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` subtree tags indicate the 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` subtree tags indicate 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` subtree tags represent things with (actual or virtual) physical existence such as objects, sounds, or language.

Descriptive tags are organized in the `Property` subtree. These descriptive tags should always be grouped with the tags they describe using parentheses.

Binary relations are in the `Relation` subtree. Like items from the `Action` subtree, these should be annotated using (`A`, (`Relation`, `B`)).

1.3.2.2 3.2.2. Tag forms

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 (/).

Valid HED tags do not have leading or trailing forward slashes (/). A HED tag path may also not have consecutive forward slashes.

An important requirement of third generation HED (versions >= 8.0.0) is that the node names in the HED schema **must be unique**. As a consequence, the user may specify as much of the path to the root as desired when using the tag in annotation.

The full path version is referred to as **long form**, and the version with only the final tag element (excluding placeholder) is called **short form**.

Any **intermediate form** of the tag path is also allowed as illustrated by this example:

Examples of allowed forms of HED tags with and without values.

Short-form	Intermediate form(s)	Long-form
Cough	*Move/Breathe/Cough Breathe/Cough*	*Action/Move/Breathe/Cough*
Weight/3 lbs	*Data-property/Data-value/Physical-value/Weight/3 lbs Data-value/Physical-value/Weight/3 lbs Physical-value/Weight/3 lbs*	*Property/Data-property/Data-value/Physical-value/Weight/3 lbs*

HED tools are available to map between shortened and long forms as needed. The tag must be associated with a schema and must correspond to a path in the schema (excluding any extension or value).

See *NODE_NAME_EMPTY* for errors involving forward slashes (/) and *TAG_INVALID* for other types of tag syntax errors.

1.3.2.3 3.2.3. Tag case-sensitivity

Although by convention tag terms start with a capital letter with the remainder being lower case, tag processing is case-insensitive. This convention makes annotation strings more readable and is recommended for tag extensions. Validators and other tools must treat tags containing the same characters, but different variations in capitalization as equivalent.

The only exception to the case-insensitive processing rule is that the correct case of unit symbols or unit modifiers should be preserved, both during schema processing and during annotation processing. This rule is required because SI distinguishes symbols and unit modifiers that differ in case.

1.3.2.4 3.2.4. Tags that take values

A HED tag that takes a value corresponds to a schema node whose unique child is a # leaf node. The actual schema `takesValue` attribute appears on the # placeholder rather than on the tag itself.

These tags may appear with or without a value. When used with a value, the tag term is followed by a slash, followed by a value.

A placeholder or its direct parent tag may not be extended in any other way. 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 substituted for the placeholder rather than as a tag extension.

If a `unitClass` is specified as an attribute of the # node, then the units specified must be valid units for that `unitClass`.

The characters that may be used in the value that replaces the # placeholder must be in the union of the values allowed by the `valueClass` attributes of the# node. If units are given, they may place additional restrictions on the allowed values.

Units with the `unitPrefix` attribute, such as $, appear before the value. Units without the `unitPrefix` attribute appear after the value. **HED parsers assume that units are separated from values by a single blank regardless of the position of the units.**

Some unit classes have the `defaultUnits` attribute specifying the units that downstream analysis tools should assume if units are omitted.

Additional checks may be made on the substituted values depending on the `valueClass`. For example `numericClass` values must be valid floating point numbers and `dateTimeClass` values must be valid ISO8601 values conforming to the **BIDS data-time requirements**.

The values of HED tag placeholders cannot stand alone, but must include the parent when used in a HED string. For example, the `Label` node in the HED schema has the # child. Thus, the value `myLabel` meant to substitute for the # child of the `Label` node must include `Label` term when used in a HED tag (e.g., `Label/myLabel` not `myLabel`).

The values substituted for # may themselves be schema node names provided they conform with any value class requirements associated with that #. Thus, Label/Item is a valid HED tag event though Item, itself, is a valid top tag. It is the Label tag with its value Item and is unrelated to the Item HED tag. However, Data-maximum/Item is not valid because the # child of Data-maximum has a valueClass=numericClass attribute and the Item value is not numeric.

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 value class.

See *VALUE_INVALID* and *UNITS_INVALID* for information on the specific validation errors associated with tags that take values.

1.3.2.5 3.2.5. Tag extensions

A tag extension, in contrast to a value, is a tag that users add as a child of an existing schema node as a more specific term for an item already in the schema. For example, a user might want to use Helicopter instead of the more general term Aircraft. Since Aircraft inherits the extensionAllowed attribute, users may use extended tags such as Aircraft/Helicopter in their annotation. The requirements for such an extension are:

Warning: Requirements for tag extensions by users:

1. Unlike values, an extension term must not already be a node in the schema.

2. The extension term contain only name characters. Blanks should be avoided.

3. The parent of the tag extension must always be included with the extended tag in annotation.

4. The extension term must satisfy the "is-a" relationship with its parent node.

5. The # placeholder cannot be used as an extension – in particular it cannot be used as a placeholder in definitions or as value annotations in sidecars.

Note: The is-a relationship is not checked by validators. It is needed so that term search works correctly.

Tag extensions should follow the same naming conventions as those for schema nodes. See *3.1.3. Naming conventions* for more information about HED naming conventions. A *STYLE_WARNING* is issued for extension tags that do not follow the HED naming convention.

Users should not use tag extension unless necessary for their application, as this breaks the commonality among annotations across datasets. Please open an **issue** proposing that the new term be added to the schema in question, if you think the term would be useful to other users.

See *TAG_EXTENSION_INVALID* for information on the specific validation errors associated invalid tag extensions.

Note: User tag extensions are sometimes accidental and due to misspelling, particularly when a long or intermediate form of the tag is used. For this reason the *TAG_EXTENDED* warning is issued for extended tags during validation.

1.3.2.6 3.2.6. Tag namespace prefixes

Users may select tags from multiple schemas, but additional schemas must be included in the HED version specification.

Users are free to use any alphabetic prefix and associate it with a specific schema in the HED version specification. Tags from the associated schema must be prefixed with this namespace designator (including the colon) when used in annotation.

Terms from only one schema can appear in the annotation without a namespace prefix followed by a colon.

See *TAG_NAMESPACE_PREFIX_INVALID* for information on the specific validation errors associated with missing schemas.

See *7.5. Library schema in BIDS* for an example of how the namespace prefix notation is used in BIDS.

1.3.2.7 3.2.7. Strings and groups

A **HED string** is an unordered, comma-separated list of HED tags and/or HED tag groups.

A **HED tag group** is an unordered, comma-separated list of HED tags and/or tag groups enclosed in parentheses. Tag groups may include other tag groups.

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

3.2.7.1. Parenthesis and order

Any ordering of HED tags and HED tag groups at the same level within a HED string is equivalent. Valid HED strings may have parentheses nested to arbitrary levels (nested groups). The parentheses must be properly nested and matched.

Parentheses are meaningful and convey association. If `A` and `B` represent HED expressions, `(A, B)` is not equivalent to the HED string `A, B`. The distinction should be preserved if possible. `(A, B)` means that HED tag `A` and HED tag `B` are associated with each other, whereas `A, B` means that `A` and `B` are each annotating some larger construct.

Specific rules of association will be encoded in a future version of the HED specification.

See *PARENTHESES_MISMATCH* for validation errors result from improper use of parentheses.

3.2.7.2. Tag group attributes

A HED tag corresponding to a schema node with the `tagGroup` attribute must appear inside parentheses (e.g., must be in HED tag group).

A HED tag corresponding to a schema node with the `topLevelTagGroup` must appear in an unnested HED group in an assembled HED annotation. Only one tag with the `topLevelTagGroup` attribute may appear in the same top-level group. The `topLevelTagGroup` attribute is usually associated with tags that have special meanings in HED such as `Definition` and `Onset`.

See *TAG_GROUP_ERROR* for information on the group errors detected based on schema attributes.

3.2.7.3. Empty tags and groups

Empty parentheses and multiple commas with no intervening tags represent empty tags and are invalid, as are HED strings with leading or trailing commas. Hence, if `A` and `B` are any HED expressions, (`A`, ((`B`))) is valid but (`A`, ()) is not.

See *TAG_EMPTY* for information on the validation errors due to empty tags or groups. Some of these errors may be reported as *COMMA_MISSING

3.2.7.4. Repeated expressions

Duplicated tag expressions at the same level in a HED tag group or HED string are not allowed. For example, the expressions (`Red`, `Blue`, `Red`) and (`Red`, `Blue`), (`Red`, `Blue`) have duplicated tag expressions at the same level and are hence invalid.

See *TAG_EXPRESSION_REPEATED* for more details on validation errors due to repeated tag expressions.

1.3.2.8 3.2.8. Special tags

3.2.8.1. The `Definition` tag

A HED definition is a tag group consist of a `Definition` tag that takes a value representing the definition's name and a tag group defining the concept. Each definition is independent and stands alone. The definition must contain a non-empty tag group.

The `Definition` tag corresponds to a schema node with the `topLevelTagGroup` attribute, assuring that definitions cannot be nested.

HED definitions may not contain any `Def` or `Def-expand` tags and must contain exactly one `Definition` tag. Multiple definitions with the same definition name are not allowed.

The `Definition` tag must be extended with a value representing the definition name and may be additionally extended by a # placeholder. If the definition name includes the # placeholder extension, then the defining tags must include exactly one tag that takes a value along with its # placeholder.

Definitions with the same name are considered duplicate definitions regardless of whether one has a placeholder and another does not. **However, each distinct substituted value represents a distinct definition name for purposes of `Onset`/`Offset` processing.**

See *DEFINITION_INVALID* for a listing of situations in which a definition may be invalid.

See also *Chapter 5.1 Creating definitions* for more details and examples.

3.2.8.2. `Def` and `Def-expand` tags

A definition is incorporated into annotations using the tag `Def/xxx` where `xxx` is the definition's name.

Alternatively, the annotator may use an expanded form (`Def-expand/xxx, yyy`) where `xxx` is the definition's name and `yyy` is a tag group containing the definitions contents.

The two usages are equivalent, and tools should be able to transform between the two representations. Note, however, that transforming from a `Def` tag to a `Def-expand` group requires the definition, while transforming from a `Def-expand` group to `Def` tag does not.

For definitions that include a placeholder, a value must be substituted for the # placeholder in `Def` tag and `Def-expand` group when final annotation assembly occurs.

See *DEF_INVALID* and *DEF_EXPAND_INVALID* for details on the types of errors that occur with `Def` and `Def-expand`.

See also *Using definitions* for more details and examples.

3.2.8.3. `Onset`, `Offset`, **and** `Inset`

The `Onset` and `Offset` tags are used to represent the temporal extent of events that have non-zero duration.

Each of these tags must appear in a top level tag group with a `Def` tag or `Def-expand` group anchor.

A tag group with an `Onset` represents the start of an event that extends over time. A tag group with an `Offset` represents the end of an event that was previously initiated by an `Onset` group. A given event of temporal extent is also terminated by the appearance of another `Onset` group with the same `Def` tag or `Def-expand` group anchor.

The `Onset` tag group may only contain its `Def` tag or `Def-expand` group anchor and at most one additional inner tag group in addition to the `Onset` tag.

The `Offset` tag group may only contain its `Def` tag or `Def-expand` group anchor in addition to the `Offset` tag.

These requirements imply that `Onset` and `Offset` must be the only tags in their tag group with the `topLevelTagGroup` attribute. `Onset` and `Offset` tags correspond to schema nodes with the `topLevelTagGroup` attribute. This implies, for example, that HED definition's contents may not include `Onset` or `Offset` tags.

An `Inset` tag designates an intermediate time point in an event of temporal extent. Like `Onset` and `Offset`, the `Inset` tag has the `topLevelTagGroup` attribute and must be anchored by a `Def` or `Def-expand`. The anchor must be the same name as that of an ongoing `Onset`. In addition to its anchor, the `Inset` tag group may contain a single additional tag group with additional information about that marked point. An event of temporal extent may contain several of these intermediate points.

See *TEMPORAL_TAG_ERROR* and *TAG_GROUP_ERROR* and for a listing of specific errors associated with onsets, and offsets, and insets.

Chapter 5.3.1 Using Onset and Offset in Chapter 5 gives examples of usage and additional details.

3.2.8.4. The `Event-context` tag

The `Event-context` tag corresponds to a schema node with both the `topLevelTagGroup` and `unique` attributes. This implies that there can be only one `Event-context` group in each assembled event-level HED string. The `Event-context` group contains information about what other events are ongoing at the time point associated with the event marker for which the annotation is included.

In general, the `Event-context` group is not included in annotations, but is generated by tools during downstream event processing.

See *TAG_GROUP_ERROR* and *TAG_NOT_UNIQUE* for additional information on validation errors related to `Event-context`.

Additional details and examples for `Event-context` can be found in *5.5. Event contexts*.

1.3.2.9 3.2.9. Sidecars

JSON sidecars are an integral part of the **BIDS** (Brain Imaging Data Structure) neuroimaging standard and are used to associate metadata with data files.

The JSON sidecars that are relevant to HED are associated with tabular data files. For example, the rows of tabular event files represent time markers on the experimental timeline, and the assembled HED annotations for each row describe what happened at that time marker. A sidecar containing annotations associated with the columns of such an event file allows HED tools to assemble HED annotations for each row of the file.

In addition to sidecars, HED annotations can also be given in the HED column of tabular files. At validation or analysis time the HED information from both the HED column of a tabular file and its associated sidecar are assembled to provide the annotation.

HED validators assume that the annotation dictionary is saved in JSON format and that they comply with the **BIDS sidecar** format.

3.2.9.1. Sidecar entries

A BIDS sidecar is a JSON dictionary with several types of entries, three of which are relevant to HED:

Three types of JSON sidecar HED-related entries.

Categorical entries

- The top-level JSON key corresponds to a column name in the event file.
- The value associated with the HED key is a dictionary of HED annotations.
- The keys of the annotation dictionary are the unique column values.
- The entry is not required to have annotations for every possible unique column value.
- Tools may choose to issue a warning if a column value does not have an annotation.
- The annotation dictionary may include annotations for values that do not appear in a particular event file.

Value entries:

- The top-level JSON key corresponds to a column name in the event file.
- The value associated with the HED key is a HED string.
- The entry's annotation is applicable to all values in its associated event file column.
- The HED annotation must contain a single # placeholder.
- Each row's column value is substituted for the # in the annotation when the row annotation is assembled.

Dummy entries:

- The top-level JSON key must not correspond to a column name in the event file.
- The value associated with the HED key is a dictionary of HED annotations.
- The keys are dummy entries.
- Used to gather HED definitions.

The other types of sidecar entries include categorical and value entries with no "HED" key, as well as arbitrary entries whose keys do not correspond to column names in an associated tabular file.

When annotations are assembled, sidecar entries with no `"HED"` key are ignored as are entries in the corresponding tabular data file that have `n/a` or blank values.

See *3.2.9.4. A sidecar example* for an elaborated example of these different types of entries and *3.2.10.2 Event-level processing* for an example of how the resulting HED annotations are assembled.

3.2.9.2. Sidecar validation

All HED-related entries in a JSON sidecar must have `"HED"` as a key in a second-level dictionary. `"HED"` cannot appear as a sidecar key that is not at the second level. Further, a sidecar is not permitted to provide a HED annotation for `n/a`. Both of these generate a *SIDECAR_INVALID* error.

HED definitions are required to be separated into dummy sidecar column entries and cannot appear in sidecar entries containing tags other than definitions. A HED definition appearing in a categorical or value sidecar entry generates a *DEFINITION_INVALID* error.

The sidecar does not have to provide a HED-relevant entry for every event file column. Columns with no corresponding sidecar entry are skipped during assembly of the HED annotation for an event file row. However, if a value is encountered in a tabular file column that is annotated as a categorical column but does not have a HED annotation, a *SIDECAR_KEY_MISSING* warning is generated.

HED value sidecar entries must contain exactly one # placeholder in the HED string annotation associated with the entry. The # placeholder should correspond to a # in the HED schema, indicating that the parent tag (also included in the annotation) expects a value. These issues generate a *PLACEHOLDER_INVALID* error.

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.

3.2.9.3. Sidecar curly braces

The curly brace notation is new with HED specification version 3.2.0 and is supported by all versions of the HED schema 8.0.0. The notation was introduced to facilitate proper nesting of HED tags associated with different event file columns when the complete HED annotation for an event marker is assembled.

When a column name appears in curly braces within a HED annotation in a JSON sidecar, the corresponding HED annotation for that row is substituted for the curly braces and their contents when the HED annotation is assembled.

Rules for curly braces notation in sidecars.

1. The item within the curly braces must either be the word `HED` or the name of another HED-annotated column within the sidecar.

2. The HED annotation for the column in curly braces directly replaces the curly braces and their contents in the target annotation.

3. During assembly of a HED annotation for an event, if the 'n/a' value appears in a curly brace column, the curly brace expression including the curly braces as well as any extra parentheses or commas are removed.

4. A sidecar column name cannot both appear in a curly braces and have an annotation that uses curly braces (to prevent circular references).

5. The curly braces cannot be used within a `Definition`.

6. Curly braces can not appear in the HED column of a tabular file.

7. Curly braces can not be nested.

8. A pair of curly braces must appear syntactically as a tag and not as the substitution for a place holder.

If curly braces appear in the HED column of a tabular file, a *CHARACTER_INVALID* error is generated.

If curly braces appear in a `Definition`, a *DEFINITION_INVALID* error is generated.

If the curly brace notation is used improperly in a sidecar or elsewhere, a *SIDECAR_BRACES_INVALID* is generated.

3.2.9.4. A sidecar example

The following example illustrates the different types of JSON sidecar entries.

Different types of sidecar annotation entries that might appear in

```json
{
    "event_type": {
        "LongName": "Event category",
        "Description": "Indicator of type of event.",
        "Levels": {
            "show": "Show a face to a participant.",
            "press": "Participant presses key to indicate symmetry."
        },
        "HED": {
            "show": "Sensory-event, Visual-presentation, {stim_file}",
            "press": "Agent-action, (Experiment-participant, (Press, {key}))"
        }
    },
    "stim_file": {
        "LongName": "Stimulus image file",
        "Description": "Time from stimulus presentation until subject presses button",
        "HED": "(Image, Face, Pathname/#)"
    },
    "key":    {
        "LongName": "Indicates which key is pressed.",
        "Description": "Indicator of participant evaluation.",
        "HED": {
            "left-arrow": "((Leftward, Arrow), Keypad-key)",
            "right-arrow": "((Rightward, Arrow), Keypad-key)"
        }
    },
    "symmetry":    {
        "LongName": "Indicates symmetrical or asymmetrical.",
        "Description": "Indicates the participant's judgement of symmetry.",
        "HED": {
            "symmetric": "(Judge, Asymmetrical)",
            "asymmetric": "(Judge, Symmetrical)"
        }
    },

    "dummy_defs": {
        "HED": {
            "MyDef1": "(Definition/Cue1, (Buzz))",
            "MyDef2": "(Definition/Image/#, (Image, Face, Label/#))"
```

```
        }
    }
}
```

In the example, "event_type" is the name of a column that is annotated using the **categorical** strategy. Its top-level dictionary has "LongName", "Description", "Levels", and "HED" keys.

The value of "Levels" is a dictionary with the unique values in the "event_type" column keyed to full text descriptions of these unique values.

The value of "HED" is a dictionary with the unique values in "event_type" keyed to the corresponding HED annotations of these unique values. In the above example, the unique values are "show" and "press". The HED annotation for show is "Sensory-event, Visual-presentation, {stim_file}".

Notice use of curly braces in the notation. Here "stim_file" must correspond to another HED-annotated column in the sidecar. The "stim_file" column is an example of a value column. Its top level dictionary keys are "LongName", "Description", and "HED". and its annotation entry: "(Image, Face, Pathname/#)". This annotation has a single #. The filename in the stim_file column replaces this # when the HED annotation for a line in an associated events.tsv file is assembled.

Since "stim_file and "key" appear within curly braces in annotations for "event_type", their HED annotations can not use curly braces.

The "dummy_defs" is an example of a **dummy annotation**. The value of this entry is a dictionary with a "HED" key pointing to a dictionary. A dummy annotation is similar in form to a **categorical annotation**, but its keys do not correspond to any event file column values. Rather it is used as a container to organize HED definitions.

In the example, Definition/Cue1 is a definition that does not use a placeholder (#) modifier in its name, while Definition/Image/# is a definition whose name Image is modified by a placeholder value. Notice that Image is both a definition name and an actual tag in the schema in this example. This is permitted.

1.3.2.10 3.2.10. Tabular files

A tabular (.tsv) file is a text file in which each line represents a row in a table. The column entries in a given row are separated by tabs. The first line of the file must contain a tab-separated list of column names, which should be unique. This description of tabular files conforms to that used by **BIDS**.

3.2.10.1 Tabular types

Generally each row in a tabular file represents an item and the columns values provide properties of that item. The most common HED-annotated tabular files represent event markers in an experiment (e.g., BIDS events.tsv files). In this case each row represents a time at which something happened.

Another common HED-annotated tabular file represents experiment participants (e.g., BIDS participants.tsv). Each row in the file represents a participant, and the columns provide characteristics or other information about the participant identified in that row.

The events.tsv files are tabular files representing markers on a timeline. This type of tabular file must have onset as the first column, and HED tools use the time values in this column to resolve Onset, Offset and Inset. In contrast, the participants.tsv file, which contains information about the experimental participants, is representative of a non time-marker file. Non time-marker files cannot use the Onset, Offset, or Inset tags as these tags are reserved for annotations of time processes and cannot be resolved if there is no time.

In any case, the general strategy for validation or other processing is:

1. Process the individual components of the HED annotation (tag-level and string-level processing).

2. Assemble the component annotations for a row (event-level or row-level processing).

3. Check consistency and relationships among the row annotations (file-level processing).

See *BIDS tabular files* for more examples.

3.2.10.2. Tabular annotations

HED annotations in tabular files can occur both in a `HED` column within the file and in an associated JSON sidecar.

The HED strings that appear in a `HED` column must be valid HED strings. If the first column is not called `onset`, the assembled annotation for the tabular file cannot contain any of the tags `Onset`, `Offset`, or `Inset`.

Definitions many not appear in the `HED` column of a tabular file or in any entry of a JSON sidecar that contains items other than definitions.

See *DEFINITION_INVALID* and *TEMPORAL_TAG_ERROR* for information.

3.2.10.3. Event-level processing

After individual HED tags and HED strings in the `HED` column of tabular files and in the associated sidecars are validated or otherwise processed, the HED strings associated with each row of the tabular file must be assembled to provide an overall annotation for the row. We refer to this as *event-level* or *row* processing.

If the HED schema used for processing contains a schema node that has the `required` attribute, then the assembled HED annotations for each row must include that tag. Currently, HED schema versions 8.0.0 do not contain any nodes with the `required` attribute, and this attribute may be deprecated in future versions of the schema.

If the HED schema used for processing contains a schema node that has the `unique` attribute, then the assembled HED annotations for each row must contain no more than one occurrence of that tag. Currently, only `Event-context` has the `unique` attribute for HED schema versions 8.0.0.

See *REQUIRED_TAG_MISSING* and *TAG_NOT_UNIQUE* for information on the validation errors that may occur with tags that have the `required` or `unique` schema attributes, respectively.

General procedure for event-level (row) assembly.

1. Create an empty result list.

2. Create an assembly list of columns that contain HED annotations and whose names do not appear in the curly braces of other HED annotations.

3. For each the column in the assembly list look up the annotation in the sidecar, replacing all curly braces and place holder values appropriately. Append to the result list.

4. If a `HED` column annotation exists for that row and `HED` did not appear in curly braces in the sidecar, concatenate the annotation to the result list.

5. Finally, join all the entries of the result list using a comma (`,`) separator.

In all cases `n/a` column values are skipped.

To illustrate the assembly process, consider the following excerpt from an event file:

General procedure for event-level (row) assembly.

---

onset	duration	event_type	stim_file	key	symmetry	HED
3.42	n/a	show	h234.bmp	n/a	n/a	"(Recording, Label/Setup)"
3.86	n/a	press	n/a	left-arrow	asymmetric	n/a
7.42	n/a	show	h734.bmp	n/a	n/a	n/a

Using the *example sidecar* results in the following assembled HED annotation for the first row of the event file:

A result for event-level (row) assembly of the sample file.

```
"Sensory-event, Visual-presentation, (Image, Face, Pathname/h234.bmp), (Recording, Label/
↪Setup)"
```

The specific annotation (`Image, Face, Pathname/h234.bmp`) has been substituted for `{stim_file}` and the annotation for in the `HED` column of the `events.tsv` file has been included. The entries with `n/a` have been ignored.

For more examples of event assembly, see **How HED works in BIDS** tutorial.

3.2.10.4. File-level processing

HED versions >= 8.0.0 allow annotation of relationships among rows in a tabular file. Hence, processing generally requires that annotations for all the rows be assembled so that consistency can be checked.

To validate temporal scope, the validator must assure that each `Onset` and `Offset` tag is associated with an appropriately defined identifier corresponding to a definition name. The validator must also check to make sure that `Onset` and `Offset` tags are properly matched within the data recording. In particular every `Offset` tag group must correspond to a preceding `Onset` tag group.

See *TEMPORAL_TAG_ERROR* for details on the type of errors that are generated due to `Onset` and `Offset` errors.

1.3.3 3.3. Semantic versioning

HED schema use the following rules for changing the *major.minor.patch* semantic version. These rules are based on the assumption that the **HED tag** short form will not require data annotators to retag their data for patch-level or minor-version changes of the schema. That is, a dataset tagged using schema version *X.Y.Z* will also validate for *X.Y+.Z+*. However, the reverse is not necessarily true. In addition, validation errors might occur during for patch-level or minor-version changes for changes or corrections in tag values or units.

Here is a summary of the types of changes that correspond to different levels of changes in the semantic version:

Change	Semantic-level
Major addition to HED functionality	Major
Tag deleted from schema.	Major
Unit or unit class removed from node.	Major
New tag added to the schema.	Minor
New attribute added to schema.	Minor
New unit class or unit added to schema.	Minor
New unit class added to node.	Minor
Node moved in schema without change in meaning.	Minor
Revision of description field in schema.	Patch
Correction of suggestedTag or relatedTag.	Patch
Correction of wiki syntax such as closing tags.	Patch

Note: It is an official policy that once in a schema, a node will not be removed. If a node becomes out-of-date, a `deprecated` attribute will be added to the tag in the schema. Suggested replacement tags should be included in the node description. A suggested replacement should be added to the tag patch table.

1.4 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 procedure 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 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 interrelationships.

1.4.1 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 in which 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 2 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.

1.4.2 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 includes a `Description` tag, 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 states 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. The 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 not addressed by the current HED specification.

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. 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.

1.4.3 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 presented*
> *in 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.

1.4.4 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 `Item` and `Agent` elements 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. `Property` 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 button*
> *to 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.)

1.4.5 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.*

1.4.6 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 computed from a program.

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 signal*
> *deviation 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 case the Blinker tool for detecting eye-blinks in EEG. The computed property is just a marker of where a feature was detected. If a value was computed at this point, an additional *Data-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. *Chapter 7: Library schemas* presents some rules for schema developers.

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.*

1.4.7 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 *Chapter 4.1:Instantaneous events*, models the sensory presentation of the stimulus images 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 *Chapter 5.3: Temporal scope*.

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 *Chapter 5.5: Event contexts*, 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.
Chapter 5.6: Experimental design describes HED mechanisms for annotating this information.

1.5 5. Advanced annotation

1.5.1 5.1. Creating definitions

HED version 8.0.0 introduced 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 *Chapter 5.3: Temporal Scope*.

A **HED definition** is a tag group that includes one `Definition` tag whose required child value is the definition's name. The definition tag group also includes an internal tag-group specifying the definition's content. The following summarizes the syntax of HED definitions.

Syntax summary for HED definitions.

Short forms:

>   *(Definition/xxx, (definition-content))*

(Definition/xxx/#, (definition-content))

Long forms:

(Property/Organizational-property/Definition/xxx, (definition-content))

(Property/Organizational-property/Definition/xxx/#, (definition-content))

Notes:

1. *xxx* is the name of the definition, and *(definition-content)* is a tag group containing the tags representing the definition's contents.

2. If the *xxx/#* form is used, then the *(definition-content)* MUST contain a single # representing a value to be substituted for when the definition is used.

The following example defines the *PlayMovie* term.

Example: *PlayMovie* defines the playing a movie on a computer 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 next example gives a definition that uses a placeholder representing a presentation rate, for example, to annotate events in which a presentation rate is varied at random. Usually the specific value substituted for the # will come from one of the columns in the `events.tsv` file.

Example: Use definition with placeholder to annotate a variable presentation rate.

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/# Hz))

Definitions may only appear in dummy entries of JSON sidecars and as external dictionaries. Definitions cannot be nested. Further, definitions must appear as top-level tag groups.

The validation checks made by the HED validator when assembling and processing definitions are summarized in *Appendix B: HED errors*. In addition to syntax checks, which occur in early processing passes, HED validators check that the definition names have unique definitions. Additional checks for temporal scope are discussed in *Chapter 5.2: Using definitions* and *Chapter 5.3: Temporal scope*.

1.5.2 5.2. Using definitions

This section describes how to use definitions to assist in annotation.

1.5.2.1 5.2.1. The *Def* tag

When a definition name such as `PlayMovie` or `PresentationRate` is used in an annotation, the name is prefixed by the `Def` tag to indicate that the name represents a defined name. In other words, `Def/PlayMovie` is shorthand for `(Visual-presentation, Movie, Computer-screen)`.

The following summarizes `Def` tag syntax rules.

Syntax summary for the Def tag:

Short forms:
> *Def/xxx*
>
> *Def/xxx/yyy*

Long forms:
> *Property/Organizational-property/Def/xxx*
>
> *Property/Organizational-property/Def/xxx/yyy*

Notes:

1. *xxx* is the name of the definition.

2. *yyy* is the value that is substituted for the definition's placeholder if it has one.

3. If the *xxx/yyy* 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 `Def` 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*

1.5.2.2 5.2.2. The *Def-expand* tag

The `Def-expand` tag provides an alternative to `Def` tag in annotations. Unlike the `Def` tag, a `Def-expand` tag must be in a tag group that includes an inner tag group with the definition's contents. If the definition includes a placeholder, that must be replaced with these contents by the appropriate value.

The following summarizes `Def-expand` tag syntax rules.

Syntax summary for the Def-expand tag:

Short forms:

> *(Def-expand/xxx, (definition-contents))*

> *(Def-expand/xxx/yyy, (definition-contents))*

Long forms:

> *(Property/Organizational-property/Def-expand/xxx, (definition-contents))*

> *(Property/Organizational-property/Def-expand/xxx/yyy, (definition-contents))*

Notes:

1. *xxx* is the name of the definition.

2. *yyy* is the replacement value for the # placeholder.

3. If the *xxx/yyy* form is used in the definition, then the replacement value (*yyy* above) must replace placeholders both in the definition's name and its contents.

The following example shows how `Def-expand` is used in an annotation.

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

Short form:

> *(Def-expand/PresentationRate/1.5 Hz,*
> *(Visual-presentation, Experimental-stimulus, Temporal-rate/1.5 Hz))*

Long form:

> *(Property/Organizational-property/Def-expand/PresentationRate/1.5 Hz,*
> *(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/1.5 Hz))*

During analysis, tools may replace `Def/PlayMovie` with a fully expanded tag string. Tools sometimes need to retain the association of the expanded tag string with the definition name for identification during searching and substitution.

1.5.3 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 distinct mechanisms for expressing temporal scope: `Onset/Offset` and `Duration/Delay`. Tools can transform between one representation and the other. However, transform from the `Duration/Delay` representation to the `Onset/Offset` representation may require the addition of additional rows (time markers) in the events file.

The mechanisms are summarized in the following table and discussed in more detail in the following sections.

Tag	Meaning	Usage
`Onset`	Marks start of event	Used with a `Def` tag or `Def-exand` group anchor. The corresponding end is marked using `Onset` or `Offset` with same anchor.
`Offset`	Marks end of event	Used with a `Def` tag or `Def-exand` group anchor. Must be preceded by an `Onset` anchored by the same definition.
`Inset`	Marks event intermediate pt	New in standard schema 8.2.0. Used with a `Def` tag or `Def-exand` group anchor. Must be within the event markersfor an `Onset` marked-event with the same anchor.
`Duration`	Marks end of an event.	Doesn't use a definition anchor.Starts at the current event marker unless `Delay`.If `Delay` included, start = current marker + delay. The offset = start + duration.
`Delay`	Marks delayed onset.	Doesn't use a definition anchor.If no `Duration`, treated as point event.Commonly for delayed response times.
`Event-context`	Context of ongoing events.	Should only be inserted by tools.Each unique event marker can have only one `Event-context` group.

All of these tags must appear in a `topLevelTagGroup`, which implies that they can't be nested. `Delay` and `Duration` will not be fully supported until HED standard schema version 8.2.0.

The `Inset` tag will also not be included until HED standard schema version 8.2.0, but is listed here for completeness.

1.5.3.1 5.3.1. Using `Onset` and `Offset`

The most direct HED method of specifying scoped events combines `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`). The `Def/xxx` is said to **anchor** the `Onset` (and similarly for `Offset`). By anchor, we mean that tools use the anchor to determine where each event of temporal extent begins and ends. A `Def-expand` tag group can also anchor the `Onset` and `Offset` groups.

The `Onset` tag group may contain an additional internal tag group in addition to the anchor `Def` tag. This internal tag group usually contains annotations specific to this instance of the event. As with all HED tags and groups, order does not matter.

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`). An intervening (`Def/xxx/zzz`, `Onset`), where `yyy` and `zzz` are different, is treated as a completely distinct temporal event.

The following table summarizes `Onset` and `Offset` usage. **Note**: A `Def-expand/xxx` tag group can be used interchangeably with the `Def/xxx`.

Syntax summary for Onset and Offset.

Short forms:

(Def/xxx, Onset, (tag-group))

(Def/xxx/yyy, Onset, (tag-group))

(Def/xxx, Offset)

(Def/xxx/yyy, 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/Organizational-property/Def/xxx, Property/Data-property/Data-marker/Temporal-marker/Offset)

(Property/Organizational-property/Def/xxx/#, Property/Data-property/Data-marker/Temporal-marker/Offset)

Notes:

1. *xxx* is the name of the definition anchoring the scoped event.

2. *yyy* is the value substituted for a definition's placeholder if it has one.

3. The *(tag-group)*, which is optional, contains tags specific to that temporal event. This tag group is not the tag group specifying the contents of the definition.

4. The additional tag-group is only in effect for that particular scoped event and not for all events anchored by *Def/xxx*.

5. If the *Def/xxx/#* form is used, the # must be replaced by an actual value.

6. 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,
(Property/Organizational-property/Def/PlayMovie,
Data-property/Data-marker/Temporal-marker/Onset,
(Property/Informational-property/Label/StarWars,
(Item/Object/Man-made-object/Media/Media-clip,
Property/Informational-property/ID/3284)))

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

[event n]
Event/Sensory-event,
(Property/Organizational-property/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 `Def/PlayMovie` "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 available to support gathering of definitions and the expansion. These definitions may be given in JSON sidecars or provided externally.

1.5.3.2 5.3.2. Using `Inset`

The `Inset` tag group marks an intermediate point in an event of temporal extent defined by `Onset` and `Offset`. Like the `Offset`, the `Inset` tag is anchored by a `Def` tag or `Def-expand` tag group that is the anchor of its enclosing `Onset`.

The `Inset` tag group may contain an additional internal tag group in addition to the anchor `Def` tag. This internal tag group usually contains annotations specific to this instance of the event. As with all HED tags and groups, order does not matter.

The following table summarizes `Inset` usage. **Note**: A `Def-expand/xxx` tag group can be used interchangeably with the `Def/xxx`.

Syntax summary for Inset.

Short forms:

> *(Def/xxx, Inset, (tag-group))*

> *(Def/xxx/yyy, Inset, (tag-group))*

Long forms:

(Property/Organizational-property/Def/xxx,
Property/Data-property/Data-marker/Temporal-marker/Inset, (tag-group))

(Property/Organizational-property/Def/xxx/#,
Property/Data-property/Data-marker/Temporal-marker/Inset, (tag-group))

Notes:

1. *xxx* is the name of the definition anchoring the scoped event.

2. *yyy* is the value substituted for a definition's placeholder if it has one.

3. The *(tag-group)*, which is optional, contains information specific to that intermediate. point in the ongoing event. This tag group is not the tag group specifying the contents of the definition..

4. The additional tag-group is only in effect at that particular point.

5. If the *Def/xxx/#* form is used, the # must be replaced by an actual value that is the same as the value used for its `Onset`.

1.5.3.3 5.3.3. Using `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 (i.e., the event offset) by adding the value of the duration to the onset of the event marker associated with that `Duration` tag. As with all HED tags and groups, order does not matter.

The following table summaries the syntax for `Duration`.

Syntax summary for Duration.

Short forms:

(Duration/xxx, (tag-group))

(Duration/xxx, Delay/yyy, (tag-group))

Long forms:

(Property/Data-property/Data-value/Spatiotemporal-value/Temporal-value/Duration/xxx,
(tag-group)

(Property/Data-property/Data-value/Spatiotemporal-value/Temporal-value/Duration/xxx, (Property/Data-
property/Data-value/Spatiotemporal-value/Temporal-value/Delay/yyy,
(tag-group))

Notes:

1. *xxx* is a time value for the duration.

2. *yyy* is a time value for the delay if given.

3. The *(tag-group)* contains the additional tags specific to the temporal event whose duration is specified.

`Duration` tags do not use a definition anchor. `Duration` should be grouped with tags representing additional information associated with the temporal scope of that event.

The `Duration` tag must appear in a top level tag group that may include an additional `Delay` tag. If the `Duration` appears with `Delay`, the end of the temporal event is the onset of the current event plus the delay value plus the duration value.

Several events with temporal-scopes defined by `Duration` tag groups may appear in the annotations associated with the same event marker.

Example: Use the Duration tag to annotate the playing of a 2-s movie clip of Star Wars.

Short form:

> *(Duration/2 s, (Sensory-event, Visual-presentation, (Movie, Label/StarWars)))*

Long form:

> *(Property/Data-value/Spatiotemporal-value/Temporal-value/Duration/2 s,*
> *(Event/Sensory-event,*
> *Property/Sensory-property/Sensory-presentation/Visual-presentation,*
> *(Item/Object/Man-made-object/Media/Visualization/Movie,*
> *Property/Informational-property/Label/StarWars)))*

The `Duration` tag has the same effect on event context as the `Onset`/`Offset` mechanism explained in *5.5. Event contexts*

The `Duration` tag is convenient because its use does not require a definition. 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 many events involving visual or auditory presentations. The use of the `Duration` tag will not be fully supported by validators until HED standard schema version 8.2.0.

1.5.3.4 5.3.4. Using `Delay`

The `Delay` tag is grouped with an inner tag group to indicate that the associated tag-group is actually an implicit event that occurs at a time offset from the current event. `Delay` tags do not use a definition anchor.

If the tag group containing the `Delay` also contains a `Duration` tag, then the tag group represents an event with temporal extent. Otherwise, it is considered a point event. As with all HED tags and groups, order does not matter.

The following table summarizes the syntax for `Delay`.

Syntax summary for Delay.

Short forms:
> *(Delay/xxx, (tag-group))*

> *(Delay/xxx, Duration/yyy, (tag-group))*

Long forms:
> *(Property/Data-property/Data-value/Spatiotemporal-value/Temporal-value/Delay/xxx,*
> *(tag-group)*

> *(Property/Data-property/Data-value/Spatiotemporal-value/Temporal-value/Duration/xxx,   (Property/Data-*
> *property/Data-value/Spatiotemporal-value/Temporal-value/Delay/yyy,*
> *(tag-group))*

Notes:

1. *xxx* is a time value for the duration.

2. *yyy* is a time value for the delay if given.

3. The *(tag-group)* contains the additional tags specific to the temporal event whose duration is specified.

A typical use case for `Delay` is when a secondary stimulus appears offset from the first. A typical use case for `Delay` combined with `Duration` is the encoding of a participant response, where the reaction time is measured relative to a secondary stimulus (such as a 'go').

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 the delay mechanism for a participant response.

Short form:

> *(Sensory-event, (Experimental-stimulus, Visual-presentation, Cross))*
> *(Delay/2.83 ms, (Agent-action, 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)),*
> *(Property/Data-property/Data-value/Spatiotemporal-value/Temporal-value/Delay/2.83 ms, (Event/Agent-action,*
> *(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))),*

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 a distinct event. For BIDS datasets, such response delays would be recorded in a column 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.

The `Delay` tag can also be used in the same top level tag group as the `Duration` tag to define an event with temporal extent. HED tools are being developed to support the expansion of delayed events to have their own event markers without the delay tag. However, use of the `Delay` tag will not be fully supported by validators until HED standard schema version 8.2.0.

1.5.4 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 o f 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` indicates that event or marker 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.

Note: Event streams are still under development.

1.5.5 5.5. Event contexts

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. Tools are available to insert the appropriate `Event-context` at analysis time. The `Event-context` has the `unique` attribute, implying that only one `Event-context` tag group may appear in the assembled HED annotation corresponding to each time-marker value.

Syntax summary for *Event-context*.

Short form:
        (Event-context, other-tag-groups)

Long form:
        (Property/Organizational-property/Event-context, other-tag-groups)

Notes:

---

1. The `Event-context` may only appear in a top-level tag group of an assembled HED string.

2. An event can have at most one `Event-context` tag group in its assembled HED annotation.

3. 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.

4. Each of these internal annotations should be in a group, indicating that they represent a distinct event process.

1.5.6 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-property` 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 may 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.

1.5.7 5.7. Specialized annotation

1.5.7.1 5.7.1. Parameter tags

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-shown*
> *Property/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.

Note: Parameters and related annotations are still under development.

1.6 6. Infrastructure and tools

The HED infrastructure includes libraries written in Python, Matlab, and JavaScript that support the use of HED in validation and/or applications. This section describes the expected behavior of the HED infrastructure and its integration into other systems such as **BIDS**.

In general, tools should either explicitly call HED validation to assure that the input tag strings are valid or should make explicit that they assume the HED has already been validated. Most tools will use the latter approach.

See *3.2. Annotation formats* for more detailed specifications of HED formats.

See *4. Basic annotation* and *5. Advanced annotation* for examples and usage.

1.6.1 6.1. Basic tag handling

HED-compliant tools should be able to a handle HED string in its equivalent forms and using various valid syntax as described in this section.

1.6.1.1 6.1.1. Tag forms

> **Warning:** HED-compliant tools should be able to handle tags in **long-form**, **short-form** or any valid **intermediate-form**.

Tools may assume that validated HED tags do not have leading, trailing, or consecutive forward slashes in their names.

In addition to being property formed, validated HED strings will correspond to terms in the schemas under which they were validated.

Tools should not distinguish between variations in case for the same tag term. Only units must have their cases preserved.

Tools may assume that the individual tags within validated HED strings have values of the proper form and that the units, if provided, are consistent with any unit classes

Note: At this time it is not required that terms with specified unit classes always have associated units. However, it is implicitly assumed that if the units are omitted in this case, the value has the default units.

See *3.2.2. Tag forms* for more information on tag forms.

1.6.1.2 6.1.2. Parentheses and commas

Tools may assume that validated HED strings have no duplicates, empty tags, empty groups (parentheses enclosing only whitespace), or mismatched parentheses.

Grouping with parentheses in HED indicates that the tags are associated. Where possible, parentheses should be preserved.

> **Warning:** HED-compliant tools should be able to handle arbitrary correctly **nested parentheses** and correctly distinguish differences in grouping.

1.6.1.3 6.1.3. Tag ordering

Any ordering of HED tags and HED tag groups at the same level within a HED tag group is equivalent.

Any ordering of top-level HED tags and HED tag groups in a HED string is equivalent.

> **Warning:** HED-compliant tools should not rely on the order that HED tags appear within a string or group during processing.

1.6.1.4 6.1.4. Definitions

> **Warning:** HED-compliant tools should be able to expand, shrink, or remove definitions.

HED definitions should only appear in sidecars in dummy entries or in an accompanying definition list. Actual `Definition` groups should not appear in the HED column of event files.

1.6.2 6.2. File-level handling

Dataset formats such as **BIDS** (Brain Imaging Data Structure) allow users to provide HED tags in multiple places. For example, BIDS dataset event files often use local codes to identify event markers in tabular (`events.tsv`) files and then provide dictionaries called JSON sidecars to map local codes to annotations.

The introduction of definitions and temporal scope for HED versions >= 8.0.0 has added additional complexity to validation and processing. Instead of being able to validate the HED string for each event individually, HED validators must now also check consistency across all events in the data-recording.

Tools should make explicit whether they support temporal scope. Tools that support temporal scope should be able to add scoped event information to the `Event-context` tag group of the intermediate events upon request.

Tools should make explicit whether they support insertion of actual events for `Delay` tag expansions and for the offsets of `Duration` tags. This information will allow analysts to call HED tools that support these operations to appropriately modify event files as a preamble to processing if the tool does not support these tags.

1.6.3 6.3. HED support of BIDS

BIDS (Brain Imaging Data Structure) is a widely-adopted specification and supporting tools for organizing and describing brain imaging and behavioral data.

1.6.3.1 6.3.1. BIDS tabular files

HED has been incorporated into the BIDS standard as the mechanism for annotating BIDS tabular files, which are tab-separated value files whose first line has the column names. A BIDS tabular file has extension `.tsv` and can be optionally accompanied by a similarly named `.json` file (i.e., a sidecar) containing metadata for the file.

The most common types of BIDS tabular files are shown in the following table.

Ends with	Time?	A row represents
`events.tsv`	Yes	Markers on the timeline of another file.
`participants.tsv`	No	Metadata about one dataset subject.
`scans.tsv`	No	Metadata about one dataset recording.
`beh.tsv`	No	A measurement not associated with an `onset`.
`samples.tsv`	No	Properties of a particular sample (e.g., tissue).
`phenotype/xxx.tsv`	No	A measurement from a particular participant.

HED treats tabular files such as (e.g., `events.tsv` files) whose first column has the name `onset` as expressing markers on a timeline. These timeline files allow `Onset`, `Inset`, and `Offset` tags in their annotations and receive additional *file-level processing* to assure that these tags are properly matched.

Tabular files that do not represent timelines are not permitted to use `Onset`, `Inset`, and `Offset` tags in their annotations. See *TEMPORAL_TAG_ERROR* for additional information.

The following shows an excerpt from a BIDS event file:

Example: Excerpt from a BIDS events.tsv file.

```
onset   duration   trial_type   response_time   HED
1.2     0.6        go           1.435           Label/Starting-point, Quiet
5.6     0.6        stop         1.739           n/a
```

The first two columns in a BIDS event file are required to be `onset` and `duration`, respectively. The `onset` is the time in seconds of an event marker relative to the start of its corresponding data recording, while the `duration` represents the duration in seconds of some aspect of the event. The remaining columns in this event file are optional.

BIDS reserves an optional column named `HED` to contain HED strings relevant for the instance represented by that row. In the above example, the first row `HED` column contains `Label/Starting-point, Quiet`, meaning that the event that starts at time 1.2 has these particular annotations in addition to annotations contributed by any relevant sidecars. The `HED` column in the second row contains `n/a`, indicating that entry should be ignored. In this case only annotations from applicable sidecars are applied.

HED annotations can also be associated with entries in other columns of the event file through an associated JSON sidecar as described in the next section.

1.6.3.2 6.3.2. BIDS timeseries

BIDS also includes another type of file with extension `.tsv` that represents continuous timeseries data. For example, the motion capture `_motion.tsv` files give samples from channels of a motion capture apparatus, while physiological data files `_physio.tsv` and `_stim.tsv` represent continuous physiological recordings such as cardiac and respiratory signals. These tabular files do not have column headers, but rather use other files to define the column names. **BIDS currently does not support HED in these types of files.**

1.6.3.3 6.3.3. 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 supports BIDS dataset format, where event metadata is contained in compatibly-named sidecars. See the *example sidecar* in Chapter 3 for an explanation of the different sidecar entries.

1.6.3.4 6.3.4. Annotation assembly

HED tools are available to assemble the annotations associated with each row in a tabular file using its HED column and the sidecar information associated with other columns of the events file.

For example, the annotations for the first row of the *example event file* above can be assembled using the *example sidecar* in Chapter 3 to give the following annotation:

Example assembled HED annotation for one event marker.

```
Sensory-event, Visual-presentation, (Square, Blue), (Delay/1.435 ms, Agent-action,
(Experiment-participant, (Press, Mouse-button))), Label/Starting-point, Quiet
```

The process is to look up the appropriate row annotation for each column in the sidecar and append these with an annotation in the HED column if available.

1.6.3.5 6.3.5. 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 or from locally cached versions when needed.

The following example `dataset_description.json` specifies that HED version 8.0.0 is used for a dataset called "A wonderful experiment".

Example: BIDS dataset description using HED version 8.0.0.

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

It is possible to include library schema in the HED version specification of the `dataset_description.json` file as shown by the following example:

Example: BIDS dataset description using HED version 8.1.0 and score library 1.0.0.

```
{
  "Name": "A great experiment",
  "BIDSVersion": "1.7.0",
  "HEDVersion": ["8.1.0", "sc:score_1.0.0"]
}
```

The version specification indicates that tags from the `score` library must be prefixed with `sc:` namespace identifier in dataset HED annotations.

The prefix notation (such as the `sc:` prefix for the `score` library in the previous example) is required when more than one schema is used in the annotation. However, namespace prefixes can be used with the standard schema as well as library schemas as illustrated by the following example.

Example: Prefixed standard schema in BIDS dataset description version specification.

```
{
  "Name": "A great experiment",
  "BIDSVersion": "1.7.0",
  "HEDVersion": ["st:8.1.0", "score_1.0.0"]
}
```

For this specification tags from the standard schema must be prefixed by `st:`, while tags from the `score` library are unprefixed. The `sc:` and `st:` namespace prefixes are arbitrary (usually short) alphabetic strings chosen by the annotation and are specific to each dataset based on its version specification.

> **Warning:** HED-compliant tools must be able to handle multiple schemas and prefixed tags.

1.6.3.6 6.3.6. HED in the BIDS validator

HED provides a JavaScript validator in the **hed-javascript** repository, which is available as an installable package via **npm**. The **BIDS validator** incorporates calls to this package to validate HED tags in BIDS datasets.

1.6.3.7 6.3.7. HED python tools

The **hedtools** package includes input functions that use **Pandas** data frames to construct internal representations of HED-annotated event files.

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 **hed-python** GitHub repository. 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.

1.7 7. Library schemas

1.7.1 7.1. Why library schemas?

The variety and complexity of events in electrophysiological experiments make 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 create pressures 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 standard HED schema unwieldy and less usable by the broader user community.

Third generation HED addressed the problem of vocabulary bloat by introducing **HED library schemas** to organize discipline-specific terminology. To use a programming analogy, when programmers write a Python module, the resulting code does not become part of the Python language or core libraries. Instead, the module becomes part of a library used in conjunction with core modules of the programming language.

A HED library schema contains the specialized vocabulary terms needed for event annotation in a specialized area. An example of such a library is the **HED SCORE schema** for annotation of EEG by clinicians.

1.7.2 7.2. Standalone schemas

1.7.3 7.3. Partnered schemas

HED library schemas were originally assumed to be **standalone** vocabularies, complete with all the needed schema attributes and properties. These standalone library schemas were usually used in conjunction with the HED standard schema, and the tags from the two different vocabularies were distinguished by prefixing the tags from one of the vocabularies with `xx:`. Here `xx:` is called the **namespace** for that schema within the annotation and is chosen by the annotator.

Partnered library schemas were introduced in HED specification version 3.2.0 and are supported by HED standard schema versions 8.2.0.

A partnered library schema version is tied to a specific version of the HED standard schema as specified in its header. A given library schema version is either **partnered** or **standalone**.

1.7.3.1 7.3.1. Partnered files

The XML file corresponding to a partnered library schema is a single, unified schema containing the information from both the library and its standard schema partner and validated as an integrated whole.

This XML merged schema file is downloaded and used by tools. Downstream tools see a single schema and can process it with no special handling. The following example shows the XML header for merged TESTLIB library version 2.0.0.

XML header for TESTLIB library 2.0.0 partnered with 8.2.0 (merged).

```
<?xml version="1.0" ?>
<HED library="testlib" version="2.0.0" withStandard="8.2.0">
```

The canonical filename for this `.xml` file is `HED_testlib_2.0.0.xml`. This file is always stored in the libraries `hedxml` directory in the **hed-schemas** GitHub repository. For the above example, the directory is **library_schemas/testlib/hedxml**.

As with any HED schema, schema builders develop and maintain their schema in MediaWiki mark-down format and use tools to convert to XML. The schema developer's version is unmerged, containing only the information specific to the library schema. The following example shows the header for the `.mediawiki` developer's version of a partnered library schema.

Mediawiki header for TESTLIB library 2.0.0 partnered with 8.2.0 (unmerged).

```
HED library="testlib" version="2.0.0" withStandard="8.2.0" unmerged="true"
```

The canonical filename for this `.mediawiki` file is `HED_testlib_2.0.0_unmerged.mediawiki`.

Tools also support an alternative form of the `.mediawiki` library schema containing all the information in the merged schema (a mirror to the XML), which may be useful for debugging, but is usually not explicitly created.

The following table summarizes the different partnered library schema formats and their uses. File names and link examples are specifically for the TESTLIB library. For other libraries, substitute the library name for the word *testlib*.

Format	Merged-status	Canonical filename	Handling
XML	merged	`HED_testlib_2.0.0.xml`	Stored in library **hedxml**.Used by tools.
XML	un-merged	`HED_testlib_2.0.0_unmerged.xml`	Can be generated but is never stored on **hed-schemas**.Not used, but available for completeness.
MediaWiki	merged	`HED_testlib_2.0.0.mediawiki`	Usually not stored in **hedwiki**.Possibly used duringschema development.
MediaWiki	un-merged	`HED_testlib_2.0.0_unmerged.mediawiki`	Working format for developersShould be stored in **hedwiki**.

1.7.3.2 7.3.2. Partnered formats

There are four significant differences between merged and unmerged MediaWiki formats:

1. The unmerged version has the `unmerged="true"` attribute in its header line.

2. The unmerged version should only include the auxiliary sections (e.g., unit classes, unit modifiers, value classes, schema attributes, and schema properties) that it explicitly extends.

3. In the unmerged schema, nodes with the `rooted=XXX` schema attribute must be top-level tags, and `XXX` must correspond to a node in the standard schema. In the merged schema, nodes with the `rooted=XXX` schema attribute are placed directly under the standard schema node `XXX`.

4. Nodes in the unmerged version cannot have the `inLibrary` attribute. In contrast, nodes from the library schema are given the `inLibrary==YYY` attribute during the merging process. Here YYY is the library schema name.

The following excerpt from an unmerged TESTLIB library schema in MediaWiki format shows a library schema node (`Data-mode`) rooted to `Statistical-value` in the standard schema.

Example of a rooted node in an unmerged schema in MediaWiki format.

```
                    . . .
'''Data-mode''' <nowiki>{rooted=Statistical-value}[A value that occurs most often in␣
↪data.]</nowiki>
* <nowiki># {takesValue, valueClass=numericClass}</nowiki>
                    . . .
```

Notice that the indentation asterisks (*) indicate that the node's children are at the first level. In the merged schema, these are adjusted accordingly as shown in the following:

When merged with the standard schema, the indentation levels are adjusted.

```
                    . . .
*** Statistical-value <nowiki>{extensionAllowed}[A value based on or employing the␣
↪principles of statistics.]</nowiki>
                    . . .
**** Data-minimum <nowiki>[The smallest possible quantity.]</nowiki>
***** <nowiki># {takesValue, valueClass=numericClass}</nowiki>
**** Data-mode''' <nowiki>{inLibrary=testlib, rooted}[A value that occurs most often in␣
↪data.]</nowiki>
***** <nowiki># {takesValue, valueClass=numericClass, inLibrary=testlib}</nowiki>
**** Probability <nowiki> [A measure of the expectation of the occurrence of a␣
↪particular event.]</nowiki>
***** <nowiki># {takesValue, valueClass=numericClass}</nowiki>
                    . . .
```

Similar differences occur between the merged and unmerged XML formats, but only the merged XML format is useful.

1.7.3.3 7.3.3. Auxiliary sections

The unmerged version of a partnered library schema **must** have prologue and epilogue sections that appropriately explain the purpose of the library schema. The contents of these prologue and epilogue sections become the prologue and epilogue, respectively, in the merged schema.

All the other auxiliary sections of the corresponding partner standard schema are inherited by the merged schema. Most unmerged partnered library schemas will not contain any additional auxiliary sections.

Auxiliary section items that do not appear in a standard schema are unlikely to be supported by the HED infrastructure if they require special handling. Thus, adding items to the auxiliary library schema sections is discouraged.

Library schema developers who need to add an item, such as a unit class to an auxiliary section, should first contact the HED Working Group to determine whether this item could be appropriately added to the standard schema. If a new item must be added, only that item and its corresponding auxiliary section should appear in the unmerged schema.

Library schema additions of units, unit classes, unit modifiers, value classes, and schema attributes are permitted, though not encouraged. **Library schemas cannot add information to the property definitions section of the schema.**

1.7.3.4 7.3.4. Partnered attributes

To support partnered library schema the following items were introduced in HED standard schema 8.2.0:

Name	Type	Role
`withStandard`	Header attribute	Indicates that this is a partnered library schema.Its value is the version of its standard schema partner.
`unmerged`	Header attribute	Indicates that this schema contains only library information.Its value is either "true" or "false.If "false", the attribute should be omitted.
`inLibrary`	Element attribute	Indicates that this element is in the library schema.Its value is the library name in lowercase.The attribute appears only in merged schemas.
`rooted=XXX`	Node attribute	Indicates that this node is to appear directly under standard schema node `XXX` in the merged schema.A node with the `rooted` attribute must bea top-level node in the unmerged schema.
`reserved`	Node attribute	Indicates that this node has special meaning or function.**Can only appear in standard schemas.**.

1.7.3.5 7.2.5. Motivation for partners

Starting with HED specification version 3.2.0 and HED standard schema version 8.2.0, **partnered library schemas** have become the recommended form for library schemas. This section describes the motivation for this preference.

7.3.5.1. Auxiliary consistency

A standalone library schema must duplicate the **auxiliary schema sections** appearing in standard schemas, introducing the possibility of inconsistency in usage or definition between the library schema and standard schemas.

Partnered library schema automatically inherit the partner standard schema's auxiliary attributes, this assuring consistent handling by tools and preventing the introduction of inconsistently handled attributes.

Although standalone library schemas may add additional items to the auxiliary sections, HED tools only guarantee support of standard schema auxiliary items requiring special handling. **Thus, addition of items in the auxiliary sections of a library schema is discouraged.**

7.3.5.2. Reserved tag handling

Several tags in the standard schema such as `Definition`, `Onset`, and `Offset` define the structure of events and the data. By partnering with a standard schema, a library schema is assured of having HED support for key features such as events of temporal extent and definitions.

Developers of partnered library schemas should release new versions whenever HED updates its standard schema. This ensures that the partnered library schema benefits from the latest updates to HED features and tools.

If the update can be done without conflict, this update may be initiated as part of the release mechanism by the maintainers of the HED repositories.

7.3.5.3. Annotation conciseness

The most common use case for library schemas in annotation requires tags from both a standard schema and a library schema, thus requiring that a `xx:` be assigned to tags from one of the schemas when standalone library schemas are used.

Because a partnered library schema is merged with a standard schema to form a single, unified schema, users can annotate data without the `xx:` namespace designator. The `xx:` is still needed if more than one library schema is used.

7.3.5.4. Library searches

The subtrees appearing in the library schemas are often elaborations of a particular term in the standard schema. However, if the library schema terms are not in appropriate standard schema hierarchy, HED search can not be leveraged to find these elaborations by searching for a more general standard schema term.

7.3.5.5. Suggested tags

Standalone library schemas cannot use the `suggestedTag` or `relatedTag` attributes to suggest using particular tags from the standard schema, since the values of the tags must be in the schemas themselves. However, with partnered library schemas, validation is only performed on the merged versions of the schema, so tags from the standard schema can be used as `suggestTag` or `relatedTag` values.

7.3.6 Lazy partnering

HED allows multiple partnered schemas to be loaded and used without prefixes provided that there are no conflicts. We refer to this process as **lazy merging**. Conflicting schemas can always be used together if all but one have an associated prefix. A merge is attempted for all non-prefixed schemas and for each group of schemas with the same prefix.

In the following example, all the library schemas are partnered with '8.2.0'. Library schemas `liba_1.0.0` and `libc_4.3.2` are merged with no prefix, and library schemas `ac:libb_2.8.1` and `ac:exam_2.3.2` are merged with prefix `ac:`. The schema `sc:test_1.3.2` stays the same and schema `8.2.0` has no effect, since it is already included as a partner of `liba_1.0.0` and `libc_4.3.2`. If there are any conflicts during the merging process, an error is raised.

Example: Merging of multiple schemas.

```
    ['liba_1.0.0', 'ac:libb_2.8.1', 'libc_4.3.2', '8.2.0', sc:test_1.3.2', 'ac:exam_2.3.
↪2']
```

Rules for lazy merging of multiple partnered schemas.

1. Partnered library schemas MUST have same standard schema partner to merge.

2. Partnered library schemas with no prefix form a merge group.

3. Schemas with the same namespace prefix form a merge group.

4. Schemas in the same merged group are merged in the order.

5. Standard schemas in a merge group are ignored if already the group partner.

6. Standard schemas in a merge group raise an error if different from the group partner.

7. The prefixes of the resulting merge groups must be unique.

---

8. If any tags match in two library schemas in a merge group, even if identical, the load fails.

9. The prologues and epilogues of the schemas are ignored since merge groups are never saved.

10. Partnered library schemas can specify schema attributes or properties.

11. New library schema unit classes and their accompanying units are merged directly.

12. New library schema units under an existing unit class are merged if no conflicts.

13. New library schema value classes are merged if no conflicts.

If an incompatible list of schemas is given, a *SCHEMA_LOAD_FAILED* error is generated.

Avoid new auxiliary section entries in library schemas.

Note: With the possible (and rare) exception of new `unitClasses` and `units`, partnered library schemas should not have auxiliary sections except for the `prologue` and `epilogue`.

Auxiliary sections have information for HED tools, and new entries may require modification to schema validation tools.

If a new entry is needed, contact the HED Working Group (hed.maintainers@gmail.com) to see if the entry might be added to the standard schema instead.

1.7.4 7.4. Library schema design

Library schema should be developed and maintained in MediaWiki format for readability. Developers should always validate the schema before converting to XML. Only validated versions of the schema should be uploaded to the GitHub **hed-schemas** repository. More information about the development process is contained in the **HED schema developers guide**.

1.7.4.1 7.4.1. General design rules

This section summarizes the general design rules for all library schema.

General design rules for HED library schema.

1. **Follow naming conventions**: A library schema must be given a name containing only alphabetic characters. This name must appear in the schema header line in the required format.

2. **Use semantic versioning**: A library library must use semantic versioning and follow the versioning update rules used by the HED standard schema as specified in *Semantic versioning*.

3. **Tag uniqueness**: Every term must be unique within the library schema and must conform to the rules for HED schema terms.

4. **Have a meaningful prologue**: The schema should include a prologue section giving an overview, purpose and scope of the library schema.

5. **Have a meaningful epilogue**: The schema should include an epilogue section containing reference, citation, and license information.

6. **Be understandable**: Schema terms should be readily understood by most users. The terms should not be ambiguous and should be meaningful in themselves without reference to their position in the schema hierarchy.

7. **Be well-organized**: If possible, no schema sub-tree should have more than 7 direct subordinate sub-trees.

8. **Maintain subtree orthogonality**: Terms that are used independently of one another should be in different subtrees (orthogonality).

9. **Enforce is-a relationship between child nodes and their parents**: Every node in a HED hierarchy must be a subclass of its parent node. This is required for HED search generalizability.

Rules 1 through 5 are enforced by validators, while rules 6 through 9 are the responsibility of the schema designers and review committees.

In general, library schema developers should avoid adding schema terms that duplicate those found in the latest HED standard schema at the time of release. Library schema developers should also try to avoid overlap of terms found in other schema libraries.

All HED schemas, including library schemas, must use **semantic versions** and adhere to the rules specified *3.3 Semantic versioning*.

Standalone library schema developers must include the auxiliary schema classes from the standard HED schema including the schema attributes, unit classes, unit modifiers, value classes, and schema properties. No changes should be made to these sections since HED tools support the special auxiliary classes from the standard schema, but in general do not support special handling of added classes beyond basic verification.

If your application requires schema classes that are not available in the standard HED schema and would like these classes to be supported, please make a request using the **issues** forum of the **hed-schemas** GitHub repository.

1.7.4.2 7.4.2. Standalone design rules

The following design rules are specifically meant for standalone library schemas.

Design rules specific to standalone HED library schemas.

1. **Avoid tag duplication**: The terms in the library schema should not overlap terms present in the latest version of the HED schema at the time of its release.

2. **Do not modify the special auxiliary sections**: The standalone library schema should exactly duplicate of special auxiliary sections of the HED standard schema that was the latest version when this schema version was released. The special sections include: schema attributes, unit classes, unit modifiers, value classes, and schema properties.

3. **Avoid adding special auxiliary items**: A library schema may not modify any of the items in the special sections of the HED standard schema.

4. **Obtain the appropriate reviews early**: Any additions to the special sections must be reviewed by the HED Working Group to determine what requirements the additions would impose on downstream tools. This should be done as early in the process as possible.

Standalone library schemas are no longer recommended because of the difficulty in enforcing conflict rules with HED standard schemas.

1.7.4.3 7.4.3. Partnered design rules

Partnered library schemas are now the recommended format for the reasons listed in *Motivation for partners*. The following design rules are specifically meant for partnered library schemas.

Design rules specific to partnered HED library schemas.

1. **Check for overlap**: The terms in the partnered library schema must not overlap with terms present in its partnered standard schema.

2. **Use the latest released version of the standard schema**: A partnered library schema should always use the latest version of the HED schema available at the time of its release.

3. **Do not put any auxiliary sections**: A partnered library schema should not contain the special auxiliary sections (e.g., schema attributes, unit classes, unit modifiers, value classes, and schema properties), unless a new item is added to the section, in which only that item should appear.

4. **Seek reviews early in the process**: Any additions to the special sections must be reviewed by the HED Working Group to determine what requirements the additions would impose on downstream tools.

It is recognized that HED standard and library schemas will both evolve and that additions or tag reorganizations may cause conflicts. These conflicts must be resolved as they occur. In general the standard schema takes precedence over any library schema in resolving these conflicts.

1.7.4.4 7.4.4. Schema namespaces

As part of the HED annotation process, users must associate one or more HED schemas with their datasets. Since it would be impossible to avoid naming conflicts across schema libraries built in parallel by different user communities, HED supports schema library namespaces to facilitate the use of multiple schemas in annotating a datasets.

If multiple schemas are used, users must define a local namespace for each additional schema and prefix the tags from each of these additional schemas by their respective namespace in annotations. The local names should be strictly alphabetic with no blanks or punctuation. If a tag namespace prefix is invalid in the version specification, a schema loading error occurs.

Example: Driving library schema example tags.

```
dp:Drive-action/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.

The introduction of partnered library schemas has greatly reduced the need for namespaces, since the most common use case is a library schema used with a standard schema.

1.7.5 7.5. Library schemas in BIDS

The most common use case (for 99.9% of the HED users) is to tag events using a standard HED schema (preferably the latest one) available in the `standard_schema/hedxml` directory of the `hed-schemas` repository of the `hed-standard` organization on GitHub. The standard schemas are available at: **https://github.com/hed-standard/hed-schemas/tree/main/standard_schema**.

The **official library schemas** are available at **https://github.com/hed-standard/hed-schemas/tree/main/library_schemas**.

Standard schemas are referenced by their version number (e.g., `8.1.0`), while library schema are referenced by a combination of library name and version number (e.g., `score_1.0.0`).

For BIDS datasets, the versions of the HED schema are specified by the `HEDVersion` field of the BIDS `dataset_description.json` file. The following example specifies that version 8.1.0 of the standard HED schema is to be used in addition to `score` library schema version `1.0.0`.

Illustration of using the namespace prefix for tagging.

The `dataset_description.json` file contains:

```
{
  "Name": "A great experiment",
  "BIDSVersion": "1.8.0",
  "HEDVersion": ["8.1.0", "sc:score_1.0.0"]
}
```

A typical annotation is:

```
"Data-feature, sc:Photomyogenic-response, sc:Wicket-spikes"
```

Based on the above description tools will download:

1. The standard HED schema:
   https://raw.githubusercontent.com/hed-standard/hed-schemas/main/standard_schema/hedxml/HED8.1.0.xml.

2. The HED `score` library schema version 1.0.0:
   https://raw.githubusercontent.com/hed-standard/hed-schemas/main/library_schemas/score/hedxml/HED_score_1.0.0.xml.

In the dataset annotations for the above example, tags drawn from the score schema would be prefixed with `sc:`, where `sc` is a local name used to distinguish tags from the additional schema.

The array specification of the schema versions in BIDS can have at most one version appearing without a colon prefix.

SCORE version 1.0.0 is not partnered, so the HED version specification had to include both the library and standard schema versions. In contrast, SCORE version 1.1.0 is partnered with HED standard schema 8.2.0, so no namespace prefixes are needed as shown in the following example:

Example: An example specification of HED version for a partnered schema.

The `dataset_description.json` file contains:

```
{
  "Name": "A great experiment",
  "BIDSVersion": "1.8.0",
```

```
  "HEDVersion": "score_1.1.0"
}
```

A typical annotation is:

```
"Data-feature, Photomyogenic-response, Wicket-spikes"
```

1.8 8. The HED ontology

———DRAFT DRAFT DRAFT DRAFT DRAFT ————

This chapter defines the HED ontology and explains the relationship between the HED ontology and the HED schema.

The `hed:` prefix is the abbreviation for HED's **IRI** (International Resource Identifier) prefix. Terms with this prefix are associated with a particular schema – either the standard schema or a library schema. Terms with the `heds:` prefix are structural elements common to all HED schemas. Both the `hed:` and `heds:` namespaces map to the same PURL (Persistent Uniform Resource Locator) as explained in *8.3 HED global identifiers*.

Terms with the `rdfs:` prefix are from the **RDF Schema** (i.e., the Resource Description Framework Schema). Terms with the `dc:` prefix are drawn from the **DublinCore Ontology**.

Finally, the GitHub source repository for the HED ontologies is **hed-ontology**. The GitHub source repository for the HED schemas is **hed-schemas**.

1.8.1 8.1. HED views and representations

1.8.1.1 8.1.1. The annotator's view

HED (Hierarchical Event Descriptors) has a standard hierarchically-organized vocabulary (the HED standard schema) and additional community-specific vocabularies (HED library schemas) that can be used to annotate and analyze experimental data. The HED vocabularies and the supporting HED ecosystem are designed to support these activities.

The data annotator's natural view is top-down – identifying a general category and then filling out the details with more specific tags in that category or grouping with descriptive tags from other categories. As illustrated in the following diagram, the HED schema hierarchy is organized in a top-down manner to support this work-flow.
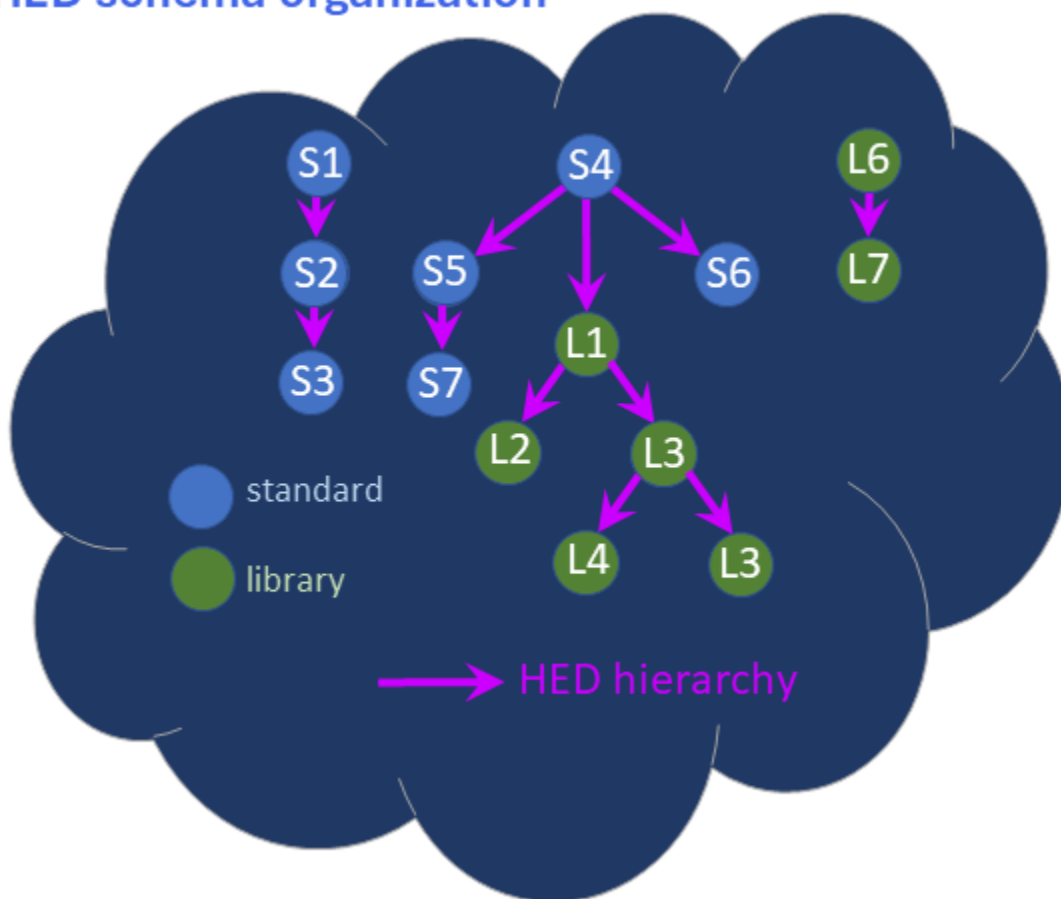
In the diagram, top-level tags `S1`, `S4`, and `L6` represent general categories and are roots of subtrees organized so that child nodes are more specific terms. HED supports "search generality", so searches may specify an exact match or matches any term in a particular subtree. In the latter case a search for `Event` may also match `Sensory-event` or `Agent-action` or any other descendent of `Event`.

The **HED schema viewer**, allows users to focus on top-level categories or expand the hierarchy view to any specified level of detail.

1.8.1.2 8.1.2. The ontologist's view

A second view of HED — the HED ontology — provides a mapping between HED schemas and classical ontologies in order to support semantic analysis and reasoning.

The following diagram shows the ontology view of HED. The nodes of the HED schema are embedded in the HED ontology, but the view is "term-centric" or "bottom-up" with links from the given term to its parent class (yellow arrows) and as well links between the given term and other terms (gray arrows).

HED ontology organization
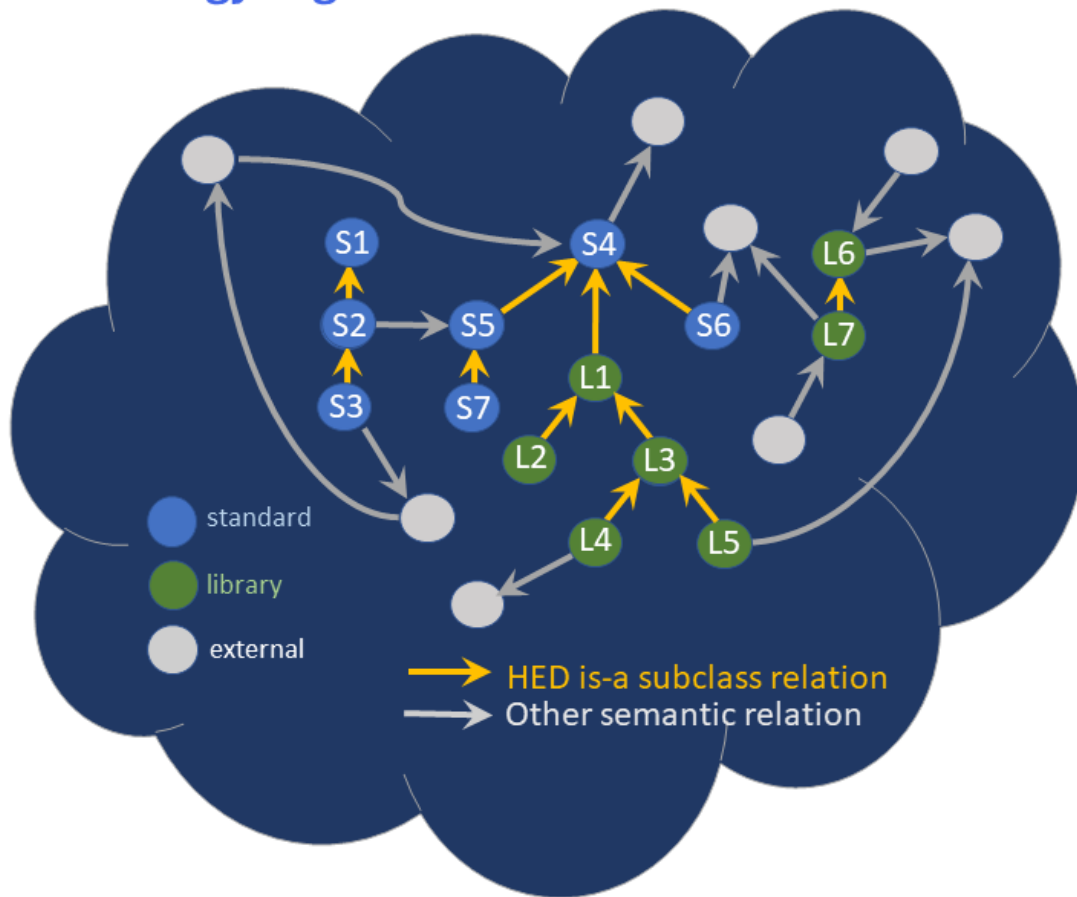
The ontology represents a complex network of interrelationships among the terms in the HED hierarchy and terms in other ontologies. The HED ontology and its mapping has been made explicit starting with HED standard schema 8.3.0. The goal is to include links to additional information including provenance and examples during annotation and to leverage AI tools during annotation and analysis.

1.8.1.3 8.1.3. HED information space

The HED schema is embedded in a larger information space that includes additional information such as sources, provenance, and links to other ontologies. This HED information space is illustrated schematically in the following diagram.
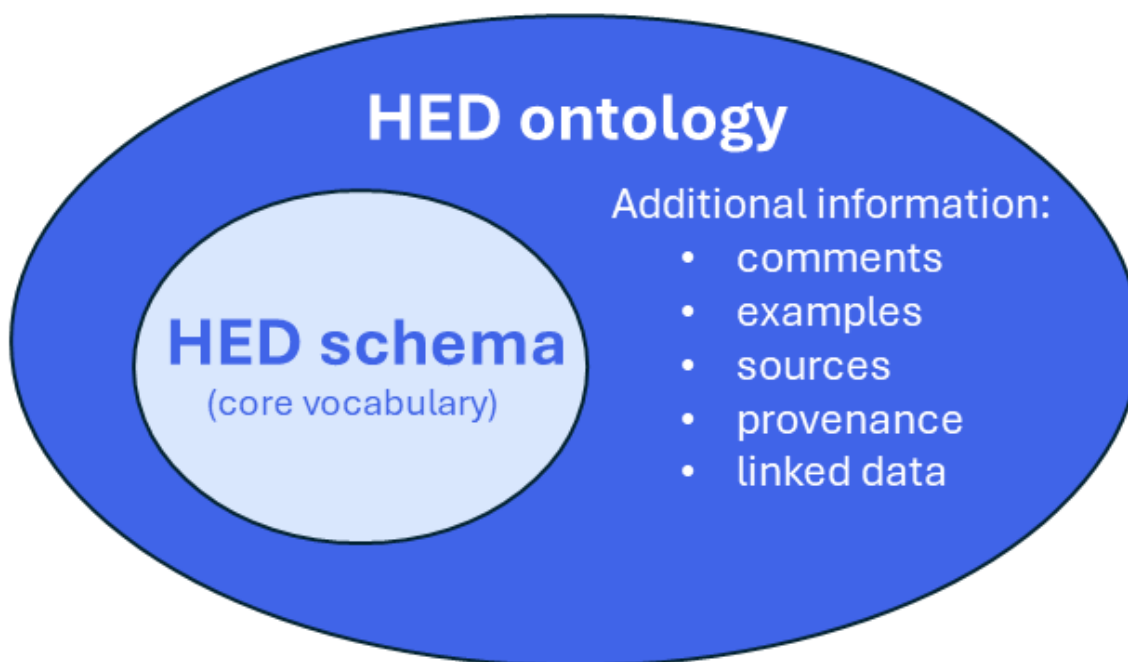
The embedding is anchored by the `hedId` schema attribute introduced with HED standard schema 8.3.0. The `hedId` values are of the form `HED_xxxxxxx` and resolve to IRIs (International Resource Identifiers) of the form *https://purl.org/hed/HED_xxxxxxx*.

The extended information space is completely represented by the HED ontology in OWL format. In this document we use OWL Manchester format (`.omn`) for readability.

1.8.1.4 8.1.4. HED representations

The HED vocabularies have 4 different formats as shown in the following table:

Format	Content	Uses	Editing
Mediawiki	Schema	Schema developmentSchema updating	Manual editingUpdated from spreadsheet
XML	Schema	Annotation toolsValidation toolsAnalysis tools	Generated from MediaWiki
Spreadsheet	Complete	Schema developmentSchema updatingOntology updating	Manual editingUpdated from MediaWikiUpdated from OWL
OWL	Complete	Ontology updatingSemantic validationDocumentation generationSemantic tools	Manual editingUpdated from spreadsheet

The HED schema, which contains the core HED vocabulary and captures the "annotator's view" of HED, is completely represented in either the HED MediaWiki and HED XML formats. The HED schema is used for annotation, validation, searching, and most analyses. Tools access the HED schema in its XML format, but schema developers create and update the schema in MediaWiki format, which is easier to read and displays as formatted MarkDown on GitHub.

The HED ontology represents the entire HED information space in **OWL** (Web Ontology Language). The HED ontology is available in both **OWL/RDF** format and **OWL Manchester** format but the examples in this specification use OWL Manchester format for readability.

8.1.4.1. Spreadsheet ⟨–⟩ Mediawiki

The HED schema also has a HED spreadsheet representation (with columns containing a tag's name, parent, attributes, and description). The mapping between the HED MediaWiki format and the HED spreadsheet is illustrated in the following example:

HED spreadsheet ↔ MediaWiki mapping
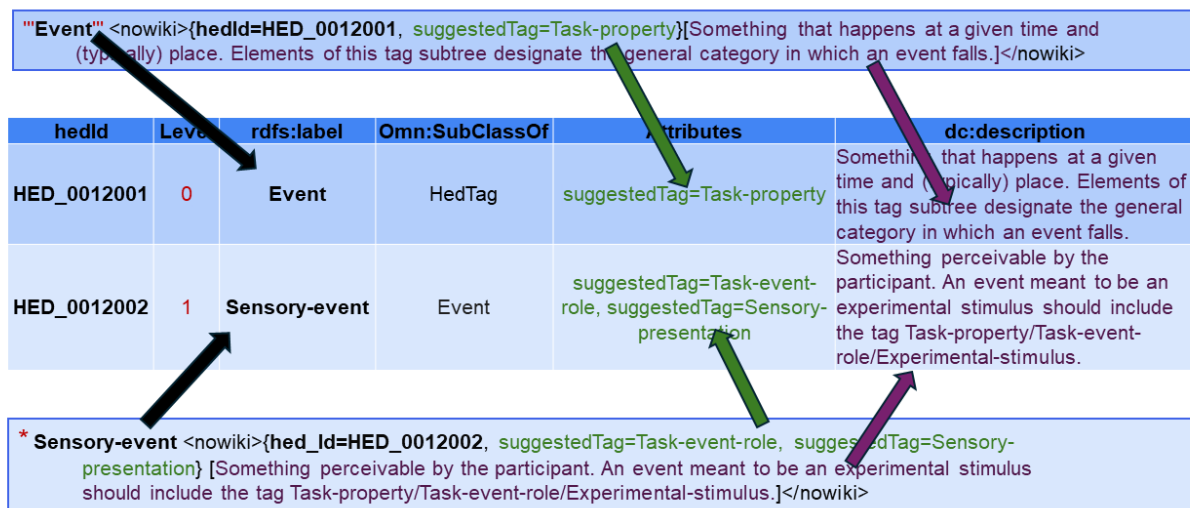
Note: The information about a single tag in the MediaWiki file must be on a single line, but the above diagram has formatted the information so that it will fit within image boundaries.

The mapping of the spreadsheet row and column values to a line in the MediaWiki is further explained in the following table:

Spreadsheet-column	Row value	MediaWiki format
hedId	*xxx*	{hedId=*xxx*, … other attributes in comma separated list}
Level	0	This row corresponds to a top-level HED tag. **EX:** `'''tag'''`.
Level	*n*	For level *n* > 0, *n* asterisks appear in front of `tag`. Example: `* tag` when *n* = 1. **EX:** `** tag` when *n*=2.
rdfs:label	*yyy*	The tag's unique name is *yyy*.
omn:SubClassOf	*zzz*	The tag's parent tag in the HED hierarchy is *zzz*.
Attributes	*uuu*, *vvv*, …	List appearing in curly braces: {hedId=*xxx*, *uuu*, *vvv*, … }
dc:description	*www*	The contents of the square braces: [*www*].

HED schema developers can develop in either HED MediaWiki or HED spreadsheet (tab-separated-value) file format and can use tools to update the representations. Users may NOT assign or modify the `hedId` in the MediaWiki file. If

adding a new tag to the MediaWiki file, schema developers should omit the `hedId`. Tools will assign and validate the `hedId` during the update process.

8.1.4.2. Spreadsheet <–> Ontology

HED spreadsheets have been expanded to include additional columns to capture all the information contained in the HED ontology as illustrated in the following example. Notice that while the spreadsheet always refers to the entities by name (e.g., `rdfs:label`), the ontology uses the `hedId`.

HED ontology ↔ spreadsheet ↔ MediaWiki mapping
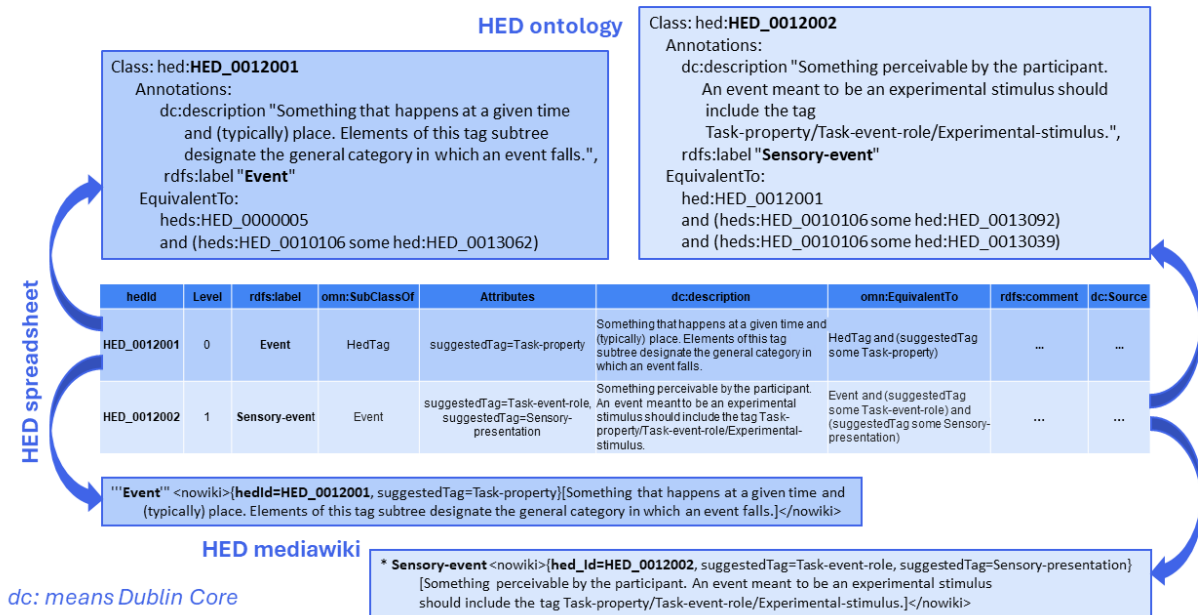
Most HED entities (i.e., tags, unit classes, units, unit modifiers, and value classes) map to classes in the HED ontology. The HED schema attributes, which describe properties and behavior, are mapped to ontology properties as described in *8.2.3. Schema attributes* section. The HED schema properties, which describe the schema attributes, determine the type of property mapping and are implicitly mapped.

The mapping of the spreadsheet row and column values to a class in the HED ontology is further explained in the following table:

Spreadsheet-column	Row value	Ontology format
hedId	*xxx*	`Class:  hed:xxx`
Level	*n*	Redundant information – can be recovered using class hierarchy.
rdfs:label	*yyy*	`rdfs:label yyy` in the `Annotations` section
omn:SubClassOf	*zzz*	The tag is either a `SubClassOf` or `EquivalentTo` *zzz* but *zzz* is identified in the ontology by **hedId** rather name.
Attributes	*uuu, vvv, …*	If non-empty, then these appear as restrictions in the `EquivalentTo`. See [**8.2.3. Schema attributes**
dc:description	*www*	`dc:description www` in the `Annotations` section.
omn:EquivalentTo		A combination of the information in **omn:SubClassOf** and **Attributes**.

The next section describes the ontology structure in more detail.

1.8.2 8.2. HED schema to ontology

Each HED element (tag, unit, unit class, unit modifier, or value class) is associated with a unique persistent globally unique identifier — the `hedId` schema attribute value in the HED schema and the entity ID in the HED ontology. The structural elements, common to all schemas also have unique assigned identifiers. The HED schema attributes are part of the HED structural ontology and

The examples in this section and `hed:` to represent schema-specific elements. Both namespaces map to the same PURL (Persistent Uniform Resource Locator).

1.8.2.1 8.2.1. Overall ontology structure

HED requires that child tags of tags in the HED schema satisfy the **is-a** relationship. This requirement is satisfied in HED ontology using subclass relationship. The HED requirement of orthogonality between tags in different top-level subtrees can be captured in the HED ontology by imposing *disjointness* on the top-level trees, but this is not currently being enforced.

The following table summarizes how the HED schema and HED ontology are mapped.

Table 2: HED ontology structure.

Schema	Ontology
Header	Class with `DataProperty` values for: • `version` - (required) semantic version of schema. • `library` - (optional) library name if library. • `withStandard` - (optional) semantic version of library's standard partner). • `merged`- (optional) library has been merged with its standard schema partner.
Tag	• Defined in the `schema` section of the HED schema. • Uses subclassing to represent HED schema structure. • Top-level tags extend `HedTag` (`heds:HED_0000005`). • A rooted library tag extends its root in the standard schema. • The parent class appears either in `SubClassOf` or `EquivalentTo`.
Unit class	Defined in the `Unit classes` section of the HED schema. • Usually only defined in the standard schema. • Defining schema must define a class extending `HedUnitClass` (`heds:HED_0000006`). • Unit classes in the standard schema are subclasses of `StandardUnitClass` (`hed:HED_0010006`).
Unit	• Defined in the `Unit classes` section of HED schema under its unit class. • Usually only defined in the standard schema. • Defining schemas must define a class extending `HedUnit` (`heds:HED_0000007`). • Units in the standard schema are subclasses of `StandardUnit` (`hed:HED_0010007`). • The class must have the `hasUnitClass` (`heds:HED_0000103`) `ObjectProperty`.
Unit modifier	• Defined in the `Unit Modifiers` section of the HED schema. • Usually only defined in the standard schema • Defining schemas must define a class extending `HedUnitModifier` (`heds:HED_0000008`). • Unit modifiers in the standard schema inherit from `StandardUnitModifier` (`HED_0010008`).
Value class	• Defined in the `Value classes` section of the HED schema. • Usually only defined in the standard schema. • Defining schemas must define a class extending `HedValueClass` (`heds:HED_0000009`). • Value classes in the standard schema inherit from `StandardValueClass` (`hed:HED_0010009`).
Schema attribute	Maps as a property in the HED ontology: • `ObjectProperty` if Range is a class and it is inherited (no `AnnotationProperty`). • `DataProperty` if Range is a literal (boolean, numeric, or string) and it is inherited. • `AnnotationProperty` if not inherited.
Schema property	• Defined in the `Properties` section of the HED schema. • Property names ending in `Domain` specify classes the attribute can be applied to. • Property names ending in `Range` specify values the property can have. • The `AnnotationProperty` indicates that this schema attribute is not inherited. • Implicitly present in the HED ontology.

1.8.2.2 8.2.2. HED Tags

The HED schema hierarchy is captured by subclassing in the HED ontology. Top-level tag nodes in the HED schema are direct subclasses of the `HedTag` class (`heds:HED_0000005`). A descendent of a top-level tag node is a direct subclass of its parent tag node in the HED schema. The ontology subclass relationship enforces the HED requirement that each tag in the HED schema must satisfy the **is-a** relationship with its parent in the HED schema.

The examples of this section use the `Action` tag and its child `Communicate` to illustrate how subclassing is represented in the various HED formats.

8.2.2.1. Mediawiki tag format

The **MediaWiki** representation uses ordering and asterisks to mark parentage relationships. In other words, the HED Wikimedia schema tree is given in depth-first search order. The parent of a tag prefixed by *X* number of asterisks is a direct child of the first tag above it with *X-1* asterisks. Top level tags are enclosed by three quotes and have no parent within the schema (i.e., *X* = 0).

For the example, the `Action` tag is a top-level tag (enclosed in a set of three quotes). The `Communicate` tag is a child (subclass) of `Action`.

Example HED MediaWiki representation of subclasses.

```
'''Action''' <nowiki>{hedId=HED_0012016}[Do something.]</nowiki>
* Communicate <nowiki>{hedId=HED_0012017}[Action conveying knowledge of or about␣
→something.]</nowiki>
```

In this example `Action` is level 0 (top-level) and `Communicate` is level 1. Mediawiki uses ordering to determine subclasses. Since `Action` is the closest preceding tag whose level is one less than that of `Communicate`, so `Action` is the parent tag of `Communicate`.

The tag's schema attributes are enclosed in curly braces, and the tag's description is enclosed in square brackets.

8.2.2.2. XML tag format

The **XML** representation of a HED tag uses nesting to indicate hierarchical relationships in the HED schema. For HED tags (nodes) the nesting indicates subclassing. For other schema elements such as unit classes and units, nesting indicates organizational grouping rather than subclasses.

Example HED MediaWiki representation of subclasses.

```xml
<node>
    <name>Action</name>
    <description>Do something.</description>
    <attribute>
        <name>extensionAllowed</name>
    </attribute>
    <attribute>
        <name>hedId</name>
        <value>HED_0012016</value>
    </attribute>
    <node>
```

```
        <name>Communicate</name>
        <description>Action conveying knowledge of or information about something.</
→description>
        <attribute>
            <name>hedId</name>
            <value>HED_0012017</value>
        </attribute>
    </node>
</node>
```

`Communicate` tag is a subclass (**is-a**) of `Action` because its <node></node> definition is nested within the <node></node> definition of `Action`.

8.2.2.3. OWL format for HED classes

We use the **OWL Manchester syntax** for the examples in this specification document because of readability. The HED ontology is also distributed in OWL/RDF format.

The following example illustrates the syntax for the HED mapping. HED schema elements (tags, unit classes, unit modifiers, units, and value classes) are mapped to OWL classes. Every element has a unique `hedId`, which is represented as an OWL `AnnotationProperty`. The `rdfs:label` annotation value is the name of the element as it appears in the HED schema. The `dc:description` annotation value is the description of the element as it appears in square brackets in the MediaWiki version of the schema.

Example HED Manchester OWL syntax.

```
Class: hed:HED_0012016
    Annotations:
        dc:description "Do something.,
        heds:hedId "HED_0012016",
        rdfs:label "Action"
    EquivalentTo:
        heds:HED_0000005
        and (heds:HED_0000102 some hed:HED_0010004)
        and (heds:HED_0010307 value "true")

Class: hed:HED_0012017
    Annotations:
        dc:description "Action conveying knowledge of or information about something.",
        heds:hedId "HED_0012017",
        rdfs:label "Communicate"
    SubClassOf:
        hed:HED_0012016
```

The `Action` (`hed:HED_0012016`) class is a top level schema tag and therefore a subclass of `HedTag` (`heds:HED_0000005`). The parentage relationship is represented by `EquivalentTo` rather than `SubClassOf` because `Action` has the `extensionAllowed` (`hed:HED_0010307`) data property.   and the `inHedSchema` (`heds:HED_0000102`) object property. Here `hed:HED_0010004` is the `HedStandardSchema` class, which has been declared elsewhere in the ontology.

The `Communicate` HED tag (`hed:HED_0012017`) is a direct child of `Action` as indicated by the `SubClassOf` entry.

Since `Communicate` is a subclass of `Action`, it inherits the `inHedSchema` association with the correct version of the standard schema.

1.8.2.3 8.2.3. Schema attributes

The purpose of HED schema attributes is to specify characteristics and/or behavior of HED schema elements, including tags, unit classes, units, unit modifiers, and value classes. These attributes map schema elements into values or into other schema elements.

8.2.3.1. Attribute ontology types

A given HED schema attribute's representation in the HED ontology is determined by its domain, its range, and whether the attribute is inherited. The mapping strategy is summarized in the following table.

Ontology	Domain	Range	Inherited?
`AnnotationProperty`	HED entity	string, numeric, boolean	No
`DataProperty`	HED entity	string, numeric, boolean	Yes
`ObjectProperty`	HED entity	HED element	Yes

`DataProperty` and `ObjectProperty` are inherited by subclasses, and reasoners can check their consistency. `AnnotationProperty` is not inherited by subclasses, and reasoners ignore them.

8.2.3.2. Schema attribute properties

Schema attribute properties appear in the `Properties` section of a HED schema. Schema attribute properties determine how schema attributes behave and described in the following table.

Attribute name	Description
`annotationProperty`	The value is not inherited by child nodes.
`boolRange`	The value can be true or false. This property was formerly named `boolProperty`.
`elementDomain`	The attribute can apply to any type of element (tag, unit, unit class, unit class, or value class). This property was formerly named `elementProperty`.
`tagDomain`	The attribute can apply to node (tag-term) elements. This property was formerly named `nodeProperty`.
`tagRange`	The value can be a node (tag).
`numericRange`	The value can be numeric.
`stringRange`	The value can be a string.
`unitClassDomain`	The attribute can apply to unit classes. This property was formerly named `unitClassProperty`.
`unitClassRange`	The value can be a unit class.
`unitModifierDomain`	The attribute can apply to unit modifiers. This property was formerly named `unitModifierProperty`.
`unitDomain`	The attribute can apply to units. This property was formerly named `unitProperty`.
`unitRange`	The value can be units.
`valueClassDomain`	The attribute can apply to value classes. This property was formerly named `valueClassProperty`.
`valueClassRange`	The value can be a value class.

Each schema attribute property is translated to an `AnnotationProperty` in the ontology.

8.2.3.3. Attribute representation

The following table lists schema attributes with their types (A=`AnnotationProperty`, D=`DataProperty`, and O=`ObjectProperty`), domains and ranges.

Attribute	Type	Domain	Range	Handling
`allowedCharacter`	D	`unitDomainunitModifierDomainvalueClassDomain`	`stringRange`	
`conversionFactor`	D	`unitDomainunitModifierDomain`	`numericRange`	
`defaultUnits`	O	`unitClassDomain`	`unitRange`	
`deprecatedFrom`	D	`elementDomain`	`stringRange`	
`extensionAllowed`	d	`tagDomain`	`boolRange`	
`hedId`	A	`elementDomain`	`stringRange`	Tools: Assign and verify.
`inLibrary`	D	`elementDomain`	`stringRange`	
`isPartOf`	O	`tagDomain`	`tagRange`	
`relatedTag`	O	`tagDomain`	`tagRange`	
`requireChild`	A	`tagDomain`	`boolRange`	Tools: Verify.
`reserved`	D	`tagDomain`	`boolRange`	
`rooted`	A	`tagDomain`	`tagRange`	Tools: make tag subclass of tag in range.
`SIUnit`	D	`unitDomain`	`boolRange`	
`SIUnitModifier`	D	`unitModifierDomain`	`boolRange`	
`SIUnitSymbolModifier`	D	`SIUnitSymbolModifier`	`boolRange`	
`suggestedTag`	O	`tagDomain`	`tagRange`	
`tagGroup`	D	`tagDomain`	`boolRange`	
`takesValue`	A	`tagDomain`	`boolRange`	Only placeholders
`topLevelTagGroup`	D	`tagDomain`	`boolRange`	
`unique`	D	`tagDomain`	`boolRange`	
`unitClass`	O	`tagDomain`	`unitRange`	
`unitPrefix`	D	`unitDomain`	`boolRange`	
`unitSymbol`	D	`unitDomain`	`boolRange`	
`valueClass`	O	`tagDomain`	`valueClassRange`	

8.2.3.4. Mediawiki attribute format

In the MediaWiki format, schema attributes appear in the `Schema Attributes` section of the schema.

Example HED MediaWiki representation of a schema attribute.

```
* extensionAllowed <nowiki>{hedId=HED_0010307,tagDomain, boolRange}[Users can add␣
→unlimited levels of child nodes
                        under this tag. This tag is propagated to child nodes except␣
→for
                        hashtag placeholders.]</nowiki>
```

Note: this example has line breaks added to fit on the page. Each element in Mediawiki must appear on one line.

The `extensionAllowed` attribute has a unique `hedId

8.2.3.5. XML attribute format

Schema attribute definitions are nested in the `<schemaAttributeDefinitions>` section of the schema's XML file. The format of an individual schema attribute is shown here.

Example HED XML representation of a schema attribute.

```
<schemaAttributeDefinition>
    <name>extensionAllowed</name>
    <description>Users can add unlimited levels of child nodes under this tag.
                This tag is propagated to child nodes except hashtag placeholders.
    </description>
    <property>
        <name>hedId</name>
        <value>HED_0010307</value>
    </property>
    <property>
        <name>tagDomain</name>
    </property>
    <property>
        <name>boolRange</name>
    </property>
</schemaAttributeDefinition>
```

8.2.3.6. OWL format for attributes.

The Manchester Owl syntax for schema attributes is similar to that of classes above.

Example HED Manchester OWL syntax for extensionAllowed.

```
DataProperty: hed:HED_0010307
    Annotations:
        dc:description "A schema attribute indicating that users can add unlimited␣
→levels of child nodes
        under this tag. This tag is propagated to child nodes except for hashtag␣
→placeholders.",
        rdfs:label "extensionAllowed"
    Domain:
        heds:HED_0000005
    Range:
        xsd:boolean
```

Unlike `DataProperty` and `ObjectProperty` attributes, `AnnotationProperty` attributes are not inherited and do not use domain and range specifications. These must be handled individual by tools as flagged by the HED schema `annotationProperty` property. In the OWL representation, information about the handling should be provided in an `rdfs:comment` field.

Example HED Manchester OWL syntax for rooted.

```
AnnotationProperty: heds:HED_0010502
    Annotations:
        dc:description "This top-level library schema node should have a parent which
        is the indicated node in the partnered standard schema.",
        rdfs:comment "Maps a HedTag into a HedTag name as a string.",
        rdfs:label "rooted"
```

1.8.2.4 8.2.4 Other auxiliary sections

The schema-ontology mapping for HED schema units, unit classes, unit modifiers, and value classes is similar to that of HED tags. However, each of these auxiliary items should inherit from a superclass tied to the schema in which they are defined.

For example the value classes that are defined in the standard schema should inherit from `StandardValueClass` (`hed:HED_0010009`) not directly from `HedValueClass` (`heds:0000009`).

1.8.3 8.3. HED global identifiers

1.8.3.1 8.3.1. Schema identifiers

The HED tags in each HED schema are unique, so a HED tag is uniquely identified by its name (label) and schema version. If the tag is from a library schema, the library name is part of the version. The rules for updating HED version numbers are specified in **HED semantic versioning**.

Starting with HED schema version 8.2.0 (released April 28, 2023), HED library schemas are strongly recommended to be *partnered with a standard schema*. Partnered schemas are joined with a specific version of the standard schema and are treated as a single integrated vocabulary for annotation and analysis. Partnered schemas MUST not have name conflicts with their standard schema partner.

Lazy partnering, introduced with HED schema version 8.3.0, allows any number of library schemas to be loaded into a single integrated vocabulary provided they are partnered with the same version of the standard schema and there are no name conflicts. If there are conflicts, user-selected namespace prefixes must be used in the version specification and in annotations to resolve the conflicts.

1.8.3.2 8.3.2. Ontology namespace

The HED ontology uses GUIDs (Global Universal Identifiers) of the form HED_xxxxxxx for all entities of HED schemas.

All HED standard and library schema entities are mapped to the `HED_xxxxxxx` namespace using the range assignments described in the following table.

HED ID	Owl type	Description
HED_0000001- HED_0000099	`Class`	HED schema structure (EX: `HedTag`).
HED_0000100- HED_0000299	`ObjectProperty`	Common object function (EX: `hasUnitClass`).
HED_0000300-HED- 0000499	`DataProperty`	Common data function (EX: `library`).
HED_0000500- HED_0000999	`AnnotationProperty`	Currently not used.
HED_0010001- HED_0010099	`Class`	Standard schema structure (EX: `StandardHeader`).
HED_0010100- HED_0010299	`ObjectProperty`	Standard schema object function (EX: `suggestedTag`).
HED_0010300-HED- 0010499	`DataProperty`	Standard schema data function (EX: `extensionAllowed`).
HED_0010500-HED- 0010699	`AnnotationProperty`	Standard schema property not inherited (EX: `takesValue`).
HED_0010700-HED- 0010899	`AnnotationProperty`	Standard schema attribute property (EX: `tagDomain`).
HED_0011300- HED_0011399	`HedValueClass`	Standard schema value class (EX:`textClass`).
HED_0011400- HED_0011499	`HedUnitModifier`	Standard schema unit modifier (EX: `deca`).
HED_0011500- HED_0011599	`HedUnitClass`	Standard schema unit class (EX: `timeUnits`).
HED_0011600- HED_0011699	`HedUnit`	Standard schema unit (EX: `hour`).
HED_0012000- HED_0039999	`HedTag`	Standard schema HED tag (EX: `Sensory-event`).
HED_0042000- HED_0059999	`HedTag`	SCORE schema HED tag (EX: `Episode`).
HED_0062000- HED_0079999	`HedTag`	LANG schema HED tag.

1.8.3.3 8.3.3. HED IRIs

HED IRIs (**International Resource Identifiers**) are mapped to **https://purl.org/hed**.

1.9 A. Schema format details

This appendix augments the discussion of HED schema formats presented in *Chapter 3: HED formats* of the HED specification. The appendix presents additional details on the rules with examples for standard HED schema and HED library schema in `.mediawiki` and `.xml` formats.

1.9.1 A.1. Auxiliary 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.

1.9.1.1 A.1.1. Unit classes and units

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.

Units corresponding to unit symbols (i.e., have a `unitSymbol` attribute) represent abbreviated versions of units and cannot be pluralized.

Elements with the `SIUnit` modifier may be prefixed with a multiple or a sub-multiple modifier. 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 `unitSymbol` 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 SI standard. However, they follow the same rules for unit modifiers as do SI units.

Table 3: 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
temperatureUnits	degree-Celsius	degree-Celsius, oC
timeUnits	s	second, s*, day, minute, hour
volumeUnits	m^3	metre^3, m^3*
weightUnits	g	gram, g*, pound, lb

1.9.1.2 A.1.2. Unit modifiers

A unit modifier can be applied to SI base units to indicate a multiple or sub-multiple of the unit. Unit symbols are modified by unit symbol modifiers, whereas SI units that are not unit symbols are modified by unit modifiers.

Table 4: SI unit modifiers

Modifier	Symbol modifier	Description
deca	da	Multiple representing 10 to power 1
hecto	h	Multiple representing 10 to power 2
kilo	k	Multiple representing 10 to power 3
mega	M	Multiple representing 10 to power 6
giga	G	Multiple representing 10 to power 9
tera	T	Multiple representing 10 to power 12
peta	P	Multiple representing 10 to power 15
exa	E	Multiple representing 10 to power 18
zetta	Z	Multiple representing 10 to power 21
yotta	Y	Multiple representing 10 to power 24
deci	d	Submultiple representing 10 to power 1
centi	c	Submultiple representing 10 to power -2
milli	m	Submultiple representing 10 to power -3
micro	u	Submultiple representing 10 to power -6
nano	n	Submultiple representing 10 to power 9
pico	p	Submultiple representing 10 to power 12
femto	f	Submultiple representing 10 to power 15
atto	a	Submultiple representing 10 to power 18
zepto	z	Submultiple representing 10 to power 21
yocto	y	Submultiple representing 10 to power 24

1.9.1.3 A.1.3. 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 extension or placeholder substitution is given a particular value class, that name or substituted value can only contain the characters allowed for by that value class.

> **Warning:** **Note**: A placeholder # specification may include multiple value class attributes.
>
> Tools check the value in question against the union of an element's `valueClass` allowed characters and any additional characters allowed by a particular unit type.

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).

Table 5: Allowed characters for value classes.

Value class	Allowed characters
dateTimeClass	`digits`, `colon`, `hyphen`, `period`, `uppercase`
nameClass	`alphanumeric`, `blank`, `hyphen`, `period`, `underscore`, `nonascii`
numericClass	`digits`, `period`, `hyphen`, `plus`, 'caret, `E`, `e`
posixPath	As yet unspecified.
textClass	`printable` or `nonascii` excluding curly braces.
IRIClass	Valid International Resource Identifier as standardized by rfc3987.

Notes on rules for allowed characters in the HED schema.

1. Commas or single quotes are not allowed in any values with the exception of the Prologue, Epilogue, term descriptions in the HED schema, and in tsv column values declared to be of type "list". The latter must be handled specially by tools.

2. Date-times should conform to ISO8601 date-time format "YYYY-MM-DDThh:mm:ss[.000000][Z]". A BIDS regular expression for this is:

```
[0-9]{4}-[0-9]{2}-[0-9]{2}T(?:2[0-3]|[01][0-9]):[0-5][0-9]:[0-5][0-9](\.[0-9]{1,6})?([A-
→Z]{2,4})?
```

3. Any variation on the full form of ISO8601 date-time is allowed.

4. The `nameClass` is for schema nodes.

5. Values of `numericClass` must be equivalent to a valid floating point value.

6. Scientific notation is supported with the `numericClass`.

7. The `textClass` is for descriptions, mainly for use with the `Description` tag or schema element descriptions.

8. The `posixPath` class is as yet unspecified and currently allows any characters except commas.

9. The IRIClass validity is determined by a library implementing the IETF rfc3987 standard.

1.9.1.4 A.1.4. Schema attributes

The type of schema element that a schema attribute may apply to is indicated by its schema type 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 6: Schema attributes.

Attribute	Domain	Range	Description
allowedCharacter	unitunit modifiervalue class	string	Specifies a character used in values of this class.
conversionFactor	unitunit modifier	numeric	Multiplicative factor to multiply by to convert to default units. (Added in version 8.1.0.)
defaultUnits	unit class	unit	Specifies units to use if placeholder value has no units.
deprecatedFrom	element	string	The latest schema version in which the element was not deprecated.
extensionAllowed	node	boolean	Users can add unlimited levels of child nodes under this tag. This tag is propagated to child nodes with the exception of the hashtag placeholders.
hedId	element	string	The unique identifier of this element in the HED namespace.
inLibrary	element	string	This schema element is from the named library schema, not the standard schema. (Added/removed by tools.)
relatedTag	node	node	A HED tag closely related to this HED tag.
requireChild	node	boolean	A child of this node must be included in the HED tag.
reserved	node	boolean	This tag has special meaning and requires special handling by tools.
rooted	node	node	A top-level library schema node should appear under this standard schema node when merged.
SIUnit	unit	boolean	This unit represents an SI unit and can be modified.
SIUnitModifier	unitModifier	boolean	Modifier applies to base units.
SIUnitSymbolModifier	unitModifier	boolean	Modifier applies to unit symbols.
suggestedTag	node	node	Tag could be included with this HED tag.
tagGroup	node	boolean	Tag can only appear inside a tag group.
takesValue	node	boolean	Placeholder (#)should be replaced by a value.
topLevelTagGroup	node	boolean	Tag (or its descendants) can be in a top-level tag group.
unique	node	boolean	Tag or its descendants can only occur once in an event-level HED string.
unitClass	node	unit class	The unit class that the value of a placeholder node can belong to.
unitPrefix	unit	boolean	Unit is a prefix (e.g., $ in the currency units).
unitSymbol	unit	boolean	An abbreviation representing a unit.
valueClass	node	value class	Type of value taken on by the value of a placeholder node.

A.1.4.1. allowedCharacter

The `allowedCharacter` attribute should appear separately for each individual character to be allowed. However, the following group designations are allowed as values for this attribute:

- `letters` designates upper and lower case alphabetic characters.

- `blank` indicates a space is an allowed character.

- `digits` indicates the digits 0-9 may be used in the value.

- `alphanumeric` indicates `letters` and `digits`

A.1.4.2. conversionFactor

The value of `conversionFactor` is the factor to multiply by the current units to convert to default units. (Added in version 8.1.0.). The `conversionFactor` value must be numeric and positive.

A.1.4.3. defaultUnits

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

A.1.4.4. deprecatedFrom

The `deprecatedFrom` attribute value takes a value that must be an existing semantic schema version earlier than the current version. Since `deprecatedFrom` has the `elementProperty`, it may be applied to any element in the schema. It is an error for the schema to use an element that has the `deprecatedFrom` attribute in a non-deprecated context. For example, if a tag has the `deprecatedFrom` attribute, then that tag may not appear as a `suggestedTag` or `relatedTag`. Deprecated schema attributes, units, unit modifiers, or value classes cannot be applied except to elements that are deprecated. In addition, the children of a deprecated tag must be deprecated or moved to a parent that is not deprecated.

A.1.4.5. extensionAllowed

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

A.1.4.6. hedId

A.1.4.7. inLibrary

A.1.4.8. relatedTag

A.1.4.9. requireChild

A.1.4.10. reserved

A.1.4.11. rooted

A.1.4.12. SIUnit

A.1.4.13. SIUnitModifier

A.1.4.14. SIUnitSymbolModifier

A.1.4.15. suggestedTag

A.1.4.16. tagGroup

A.1.4.17. takesValue

A.1.4.18. topLevelTagGroup

A.1.4.19. unique

A.1.4.20. unitClass

A.1.4.21. unitPrefix

A.1.4.22. unitSymbol

A.1.4.23. valueClass

A.1.4.x. Deprecated attributes

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 7: Schema attributes deprecated for versions 8.0.0.

Schema attribute	Target	Description
default	node	A default value used if no value is provided. Removed in standard schema version 8.0.0.
position	node	Indicates where this tag should appear during display. Removed in standard schema version 8.0.0.
predicateType	node	Indicates the relationship of the node to its parent. Removed standard schema version 8.0.0.
recommended	node	Event-level HED strings should include this tag. Removed in standard schema version 8.3.0.
required	node	Event-level HED string must include this tag. Removed in standard schema version 8.3.0.

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 were to be shown after the others in canonical ordering. The tagging strategy of HED versions >= 8.0.0 using decomposition and definitions does not permit this type of ordering. The `position` attribute is not used for HED versions >= 8.0.0.

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. In HED versions >= 8.0.0, the parent-child relationship MUST be `subclassOf` to allow search generality. The attribute is ignored by tools.

1.9.1.5 A.1.5. Schema properties

The schema properties are qualifiers on the domains and ranges of schema attributes. A property's presence implies the attribute has this property, while its absence implies it does not. Processing of `property` elements is hard-coded into the schema processors. The following is a list of schema attribute properties.

Table 8: Summary of schema attribute properties for HED Version >= 8.0.0.

Property	Description
annotationProperty	This schema attribute is NOT inherited.Replaces `isInheritedProperty`.
boolRange	This schema attribute's value can be true or false.This property was formerly named `boolProperty`.
elementDomain	This schema attribute can apply to any typeof element (tag term, unit class, etc).This property was formerly named `elementProperty`.
isInheritedProperty	**Deprecated from 8.2.0** in favor of `annotationProperty`.This schema attribute is inherited by child nodes.This property only applies to schema attributes for nodes.
tagDomain	This schema attribute can apply to node (tag-term) elements.This was added so attributes could apply to multiple types of elements.This property was formerly named `nodeProperty`.
tagRange	This schema attribute's value can be a node.This property was formerly named `nodeProperty`.
numericRange	This schema attribute's value can be numeric.
stringRange	This schema attribute's value can be a string.
unitClassDomain	This schema attribute can apply to unit classes.This property was formerly named `unitClassProperty`.
unitClassRange	This schema attribute's value can be a unit class.
unitModifierDomain	This schema attribute can apply to unit modifiers.This property was formerly named `unitModifierProperty`.
unitDomain	This schema attribute can apply to units.This property was formerly named `unitProperty`.
unitRange	This schema attribute's value can be units.
valueClassDomain	This schema attribute can apply to value classes.This property was formerly named `valueClassProperty`.
valueClassRange	This schema attribute's value can be a value class.

Property names ending in `Range` designate the type of value a schema attribute has. Starting with HED standard schema version 8.3.0 the `boolProperty`, which indicates that a schema attribute value can be true or false, was renamed `boolRange`. In addition, `numericRange` and `stringRange` were added, since the `conversionFactor` schema attribute has a numeric value.

Property names ending in `Domain` indicate the type of schema element that a schema attribute applies to. String with HED standard schema version 8.3.0 the property names `elementProperty`, `nodeProperty`, `unitClassProperty`, `unitModifierProperty`, `unitModifierProperty`, `unitProperty`, and `valueClassProperty` were renamed as `elementDomain`, `tagDomain`, `unitClassDomain`, `unitModifierDomain`, `unitModifierDomain`, `unitDomain`, and `valueClassDomain` to better clarify their role and to facilitate mapping to the HED ontology.

Format for schema attributes with schema property values.

Attributes with boolean range (`boolRange`):

- In `.xml` the attribute appears as a `<name>` element with the property's name but no `<value>` in an `<attribute>` section of the schema element.

- In `.mediawiki`, the attribute name appears in curly braces in the element's specification line.

- In either case presence of the property indicates true, and absence indicates false.

Schema without a boolean range:

- In `.xml`, the attribute appears with both `<name>` and `<value>` in an `<attribute>` section of the schema element.

- In `.mediawiki`, the schema element has the `{name =value}` in the element's specification line.

- These schema attributes may appear multiple times in an element with different values if appropriate.

See *property example* for an example in MediaWiki format and its https://hed-specification.readthedocs.io/en/latest/Appendix_A.html#a-3-5-5-schema-properties

1.9.2 A.2. 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.

1.9.2.1 A.2.1. Overall file layout

Overall layout of a HED MEDIAWIKI schema file.

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

1.9.2.2 A.2.2. MediaWiki header

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

Table 9: Allowed HED schema header parameters

Name	Level	Description
library	optional	Name of library used in XML file names. The value should only have lowercase 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 an XSD file.
withStandard	optional	The version of the standard schema partner if this is a partnered library schema.
unmerged	optional	If true, this is an unmerged partnered library schema. If omitted, assumed false.

The following example gives a sample *header* for standard schema version 8.0.0 in `.mediawiki` format.

Example: Sample *header* for version 8.0.0 in .mediawiki format.

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

The schema `.mediawiki` file specified in this example is named `HED8.0.0.mediawiki` and can be found in the **standard_schema/hedwiki** directory of the **hed-schemas** GitHub repository.

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. The `xsi` attribute is required if `xmlns:xsi` is given. The **XSD file** allows validators to check the format of the `.xml` using standard XML validators.

The following example shows a sample *header* for `testlib` library schema version 1.0.2 in `.mediawiki` format.

Example: Sample *header* for testlib library version 1.0.2 in .mediawiki format.

```
HED library="testlib" version="1.0.2"
```

The `library` and `version` values are used to form the official file name `HED_testlib_1.0.2.mediawiki`. The file is found in **library_schemas/testlib/hedwiki** directory of the **hed-schemas** GitHub repository.

A warning is generated when unknown header attributes are translated as attributes of the `HED` line during `.mediawiki` file validation.

1.9.2.3 A.2.3. Mediawiki prologue and epilogue

The prologue is an optional paragraph of text appearing after the *header*. The prologue is used by tools for help and display purposes.

Early versions of HED use the prologue section to record a CHANGE_LOG as well as information about the syntax and rules. HED versions 8.0.0 include a separate change log file for released versions.

Similar to the prologue section, the epilogue is an optional paragraph of text, usually containing references and license information. The epilogue appears directly before the ending line of the file.

Both the prologue and epilogue may contain commas and new lines in addition to the characters specified by the `textClass`.

1.9.2.4 A.2.4. Mediawiki schema section

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

```
!# start schema
```

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

```
!# end schema
```

A section separator is a line starting with !#. 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.

The body of the HED specification is located between the `!# start schema` and `!# end schema` section separators. Each specification is a single line in the `.mediawiki` file.

The three types of lines in the main specification section are **top-nodes**, **normal-nodes**, and **placeholders**, respectively.

Empty lines or lines containing only blanks are ignored.

The basic format for a node-specification is:

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

Top node names are enclosed in triple single quotes (e.g., `'''Event'''`), while other types of nodes 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. The attributes are in curly braces ({ }) and the description is in square brackets ([ ]).

Node names in HED versions 8.0.0 can only contain alphanumeric characters, hyphens, and under-bars (i.e., they must be of type `nameClass`). They cannot contain blanks and must be unique.

HED versions < 8.0.0 allow blanks in node names and also have some duplicate node names. Use of HED versions < 8.0.0 is deprecated and validators no longer support their use.

For top nodes and normal nodes, everything after the node name must be contained within <nowiki></nowiki> tags. The # is included within the <nowiki></nowiki> tags in placeholder nodes.

Example: Different types of HED node specifications in .mediawiki format.

Top node:

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

Normal node:

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

Placeholder node:

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

The `Duration` tag of this example is at the fifth level below the root (top node) 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.

1.9.2.5 A.2.5. MediaWiki auxiliary sections

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

A.2.5.1. Unit classes and units

Unit classes specify the types of units allowed to be used with a value substituted for a # placeholder.

The unit class specification section starts with `'''Unit classes'''` and lists the types of units (the unit classes) at the first level and the specific units corresponding to those unit classes at the second level.

Example: Part of the HED unit class for time in .mediawiki format.

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

A.2.5.2. Unit modifiers

The SI units can be modified by SI (International System Units) sub-multiples and multiples. All unit modifiers are at level 1 of the `.mediawiki` file.

Example: Part of the HED unit modifier in .mediawiki format.

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

A unit must have the `SIUnit` attribute in order to be used with modifiers. If the unit has both the `SIUnit` and `unitSymbol` attributes, then it only can be used with `SIUnitSymbolModifier` modifiers. If the unit has only the `SIUnit` attribute, then it only can be used with the `SIUnitModifer`.

For example the unit `second` is an `SIUnit` but not a symbol, so `second`, `seconds`, `decasecond` and `decaseconds` are all valid units.

The unit `s` is both a `SIUnit` and a `unitSymbol`, so `s` and `das` are valid units. Note that rules about pluralization do not apply to unit symbols.

A.2.5.3. Value classes

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

Example: Part of the HED value class for date-time in .mediawiki format.

```
'''Value classes'''
* dateTimeClass <nowiki>{allowedCharacter=digits,allowedCharacter=T,allowedCharacter=-,
→allowedCharacter=:}[Should conform to ISO8601 date-time format YYYY-MM-DDThh:mm:ss.]</
→nowiki>
```

A.2.5.4. Schema attributes

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: HED schema attributes allowedChaaracter and defaultUnits in .mediawiki format.

```
'''Schema attributes'''
* allowedCharacter <nowiki>{valueClassProperty}[Value may contain this character.]</
→nowiki>
* extensionAllowed <nowiki>{boolProperty}[This schema node may be extended.]</nowiki>
```

The schema attributes, themselves, have attributes referred to as *schema properties*. These schema properties are listed in the `Properties` section of the schema. The example indicates that `allowedCharacter` is associated with value classes, while `defaultUnits` is associated with unit classes.

A.2.5.5. Schema properties

Properties apply only to schema attributes. The following example defines the `valueClassProperty` in `.mediawiki` format.

Example: HED schema property valueClassProperty in .mediawiki format.

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

See *Schema properties* for a list of available schema properties.

1.9.3 A.3. XML file format

This section describes details of the XML schema format.

1.9.3.1 A.3.1. XML file layout

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
<?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>

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

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

1.9.3.2 A.3.2. XML header

The HED node is the root node of the XML schema.

Example: Header for Version 8.0.0 of the standard HED XML schema.

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

The file name corresponding to this example is HED8.0.0.xml. The file is found in the **standard_schema/hedxml** directory of the **hed-schemas** GitHub repository.

Library schemas must include the library attribute with the library name in their header line as shown in the following example.

Example: Version 1.0.2 of HED testlib library schema in .xml format.

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

The library and version values are used to form the official xml file name HED_testlib_1.0.2.xml. The file is found in **library_schemas/testlib/hedxml** directory of the **hed-schemas** GitHub repository.

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

The library and version values are used to form the official xml file name HED_testlib_1.0.2.xml.

Library schemas may also be partnered as is HED_testlib_2.0.0.xml.

Example: Partnered library schema testlib version 2.0.0 in .xml format.

```
<HED library="testlib" version="2.0.0" withStandard="8.2.0">
```

1.9.3.3 A.3.3. XML prologue and epilogue

The <prologue>...</prologue> and <epilogue>...</epilogue> elements are meant to be treated as opaque as far as schema processing goes.

HED versions < 8.0.0 contained a Change Log for the HED schema in the prologue section as well as some basic documentation of syntax. The epilogue section contained additional metadata to be ignored during processing.

1.9.3.4 A.3.4. XML 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` description 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 format of the XML attributes was changed with HED versions &ge 8.0.0. Earlier versions of the schema have been deprecated and tools no longer support their validation.

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>
```

Example: The suggestedTag is a schema attribute that has a value. The 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>
```

1.9.3.5 A.3.5. XML auxiliary sections

The auxiliary sections define various aspects of behavior of various types of elements in the schema.

A.3.5.1. Unit classes

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

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 as shown in the following example.

Example 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>
```

A.3.5.2. Unit modifiers

Unit modifiers are defined in the `<unitModifierDefinitions>` section of the XML schema file. The following shows the layout of an example unit modifier definitions:

Example XML layout of the unit modifier definition

```
<unitModifierDefinitions>
    <unitModifierDefinition>
        <name>deca</name>
        <description>SI unit multiple representing 10^1.</description>
        <attribute>
            <name>SIUnitModifier</name>
        </attribute>
        <attribute>
            <name>conversionFactor</name>
            <value>10.0</value>
        </attribute>
    </unitModifierDefinition>
            . . .
</unitModifierDefinitions>
```

A.3.5.3 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>`:

Example XML layout of the unit class definitions.

```
<valueClassDefinitions>
   <valueClassDefinition>
       <name>dateTimeClass</name>
       <description>Should conform to ISO8601 date-time format YYYY-MM-DDThh:mm:ss.</
→description>
       <attribute>
           <name>allowedCharacter</name>
           <value>digits</value>
           <value>T</value>
           <value>-</value>
           <value>:</value>
       </attribute>
   </valueClassDefinition>
</valueClassDefinitions>
```

A.3.5.4. 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.

Example XML layout of the schema attribute definitions.

```
<schemaAttributeDefinitions>
    <schemaAttributeDefinition>
        <name>allowedCharacter</name>
        <description>Value may contain this character.</description>
        <property>
            <name>valueClassProperty</name>
        </property>
    </schemaAttributeDefinition>
    <schemaAttributeDefinition>
        <name>extensionAllowed</name>
        <description>This schema node may be extended.</description>
        <property>
            <name>boolProperty</name>
        </property>
    </schemaAttributeDefinition>
    . . .
</schemaAttributeDefinitions>
```

A.3.5.5. Schema properties

The following is an example of the layout of the `valueClassProperty` in `.xml` format.

Example XML layout of the schema property definitions.

```
<propertyDefinitions>
          . . .
    <propertyDefinition>
        <name>valueClassProperty</name>
        <description>Indicates that the schema attribute is meant to be applied to␣
↪value classes.</description>
    </propertyDefinition>
</propertyDefinitions>
```

See *Schema properties* for a list of available schema properties.

1.10 B. HED errors

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

HED-compliant tools may assume that it if a HED annotation has been properly validated, it will comply with the rules of the HED specification. Annotators and analysts are mainly concerned with HED validation errors relating to incorrectly annotated events. See *B.1: HED validation errors* for a listing of errors keyed to the HED specification.

HED-compliant tools assume that the HED schemas available on the **hed-standard/hed-schemas** GitHub repository are error-free, and that schema errors can only occur due to failure to locate or read a HED schema.

HED schema developers are mainly concerned with errors and inconsistencies in the schema itself. Schemas under development should be validated at all stages of development. See *B.2: Schema validation errors* for a listing of errors keyed to the HED specification.

1.10.1 B.1. HED validation errors

1.10.1.1 CHARACTER_INVALID

A HED string contains an invalid character.

a. A tag extension contains a non `name` character.
b. Curly braces appear in a HED string not in a sidecar.

Notes:

- HED uses ANSI encoding and does not support UTF-8.

- Different parts of a HED string have different rules for acceptable characters.

See *3.2.4 Tags that take values* and *3.2.5: Tag extensions* for an explanation of the rules for tag values and extensions.

1.10.1.2 COMMA_MISSING

HED tag groups and tags must be separated with commas. In the following `A`, `B`, `C`, and `D` represent HED expressions.

a. Two tag groups are not separated by commas: `(A, B)(C, D)`.
b. A tag and a tag group are not separated by commas: `A(B,D)`.

Note: Commas missing between two HED tags are generally detected as invalid HED tags, rather than as missing commas.

See *3.2.7.3. Empty tags and groups* for an explanation of the rules for empty tags.

See also *TAG_EMPTY*.

1.10.1.3 DEF_EXPAND_INVALID

a. A `Def-expand` tag's name does not correspond to a definition.
b. A `Def-expand` is missing an expected placeholder value or has an unexpected placeholder value.
c. A `Def-expand` has a placeholder value of incorrect format or units for definition.
d. The tags within a `Def-expand` do not match the corresponding definition.
e. A `Def-expand` tag group is missing its inner tag group.
f. A `Def-expand` tag group has extra tags or groups.

See *3.2.8.2. The Def and Def-expand tags* for an explanation of the rules for `Def-expand` and *5.2. Using definition* for more details and examples.

1.10.1.4 DEF_INVALID

a. A `Def` tag's name does not correspond to a definition.
b. A `Def` tag is missing an expected placeholder value or has an unexpected placeholder value.
c. A `Def` has a placeholder value of incorrect format or units for definition.

See *3.2.8.2. The Def and Def-expand tags* for an explanation of the rules for `Def` and *5.2. Using definition* for more details and examples.

1.10.1.5 DEFINITION_INVALID

A **definition** is a tag group containing a `Definition` tag and a single tag group with the definition's contents.

a. A `Definition` tag does not appear in a tag group at the top level in an annotation.
b. A definition's enclosing tag group is missing the inner tag group (.i.e., the definition's contents).
c. A definition's enclosing tag group contains more than a `Definition` tag and an inner group.
d. A definition's inner tag group contains `Definition`, `Def` or `Def-expand` tags.
e. A definition uses curly braces.
f. A definition that includes a placeholder (#) does not have exactly two # characters.
g. A definition has placeholders (#) in incorrect positions.
h. Definitions of the same name appear with and without a #.
i. Multiple `Definition` tags with same name are encountered.
j. A tag with a `required` or `unique` attribute appears in a definition.
k. A definition appears in an unexpected place such as an events file or in a sidecar dictionary that contains non-definition entries.

See *3.2.8.1. The Definition tag* for an explanation of the rules for definitions. See also *5.1. Creating definitions* and *5.2. Using definitions* for more details and examples of definition syntax.

1.10.1.6 ELEMENT_DEPRECATED*

A schema element has been deprecated and should not be used. When an element is updated, its description is updated to include the reason for deprecation and a suggested path for updating usage.

a. A tag has been deprecated and an alternative method of tagging should be used.
b. A unit, unit class, value class has been deprecated and an alternative should be used.

See *A.1.4. Schema attributes* for additional information about the `deprecatedFrom` schema attribute.

1.10.1.7 NODE_NAME_EMPTY

a. A tag has one or more forward slashes (/) at beginning or end (ignoring whitespace).
b. A tag contains consecutive forward slashes (ignoring whitespace).

See *3.2.3 Tag forms* for more information.

1.10.1.8 PARENTHESES_MISMATCH

a. A HED string does not have the same number of open and closed parentheses.
b. The open and closed parentheses are not correctly nested in the HED string.

See *3.2.7.1. Parentheses and order* for the rules for parentheses in HED.

1.10.1.9 PLACEHOLDER_INVALID

a. A # appears in a place that it should not (such as in the `HED` column of an events file).
b. A JSON sidecar has a placeholder (#) in the HED dictionary for a categorical column.
c. A JSON sidecar does not have exactly one placeholder (#) in each HED string representing a value column.
d. A placeholder (#) is used in JSON sidecar or definition, but its parent in the schema does not have a placeholder child.

See *3.2.4. Tags that take values* and *3.2.9.1. Sidecar entries* for information on the use of placeholders in HED.

1.10.1.10 REQUIRED_TAG_MISSING

a. An event-level annotation does not have a tag corresponding to a node with the `required` schema attribute.

Note: An assembled event string must include all tags having the *required* schema attribute.

See *3.2.10.2. Event-level processing* for additional information on the `required` tag.

1.10.1.11 SCHEMA_LOAD_FAILED

a. Different standard schema versions in a merge group.
b. Library schemas in merge group have the same tag.

See [**7.2.5.6 Loading multiple partnered schemas**](./07_Library_schemas.md#7256-loading- multiple-partnered-schemas) for a more detailed description of the rules.

1.10.1.12 SIDECAR_BRACES_INVALID

a. A name appearing in curly braces in a sidecar HED annotation is not the word HED or the name of a HED-annotated column in the sidecar.
b. A column name entry in a sidecar has a HED annotation with curly braces, but this name also appears in curly braces in another HED annotation.
c. The curly braces in a sidecar are nested or unmatched.
d. The curly braces appear as the substitution for a placeholder in another tag.

See *3.2.9. Sidecars* for information on the requirements for using sidecars.

1.10.1.13 SIDECAR_INVALID

a. The "HED" key is not a second-level dictionary key.
b. An annotation entry is provided for n/a.

See *3.2.9. Sidecars* for a general explanation of sidecar requirements.

1.10.1.14 SIDECAR_KEY_MISSING*

(WARNING)

a. A value in a categorical column does not have an expected entry in a sidecar.

Note: This warning is only triggered if the categorical column in which the value appears does have HED annotations.

See *3.2.9. Sidecars* for a general explanation of sidecar requirements.

1.10.1.15 STYLE_WARNING*

(WARNING) **a.** An extension or label does not follow HED naming conventions.

See *3.1.3. Naming conventions* for an explanation of HED naming conventions.

1.10.1.16 TAG_EMPTY

a. A HED string has extra commas or parentheses separated by only white space, indicating empty tags.
b. A HED string begins or ends with a comma (ignoring white space), indicating an empty string.
c. A tag group is empty (i.e., empty parentheses are not allowed).

See *3.2.7.3. Empty tags and groups* for the rules on empty tags and groups.

1.10.1.17 TAG_EXPRESSION_REPEATED

a. A tag is repeated in the same tag group or level.

Suppose A, B, and C represent HED expressions. HED strings are not ordered, so (B, C) is equivalent to (B, C). Thus, (A, (A, B)) is not a duplicate, but (A, (B, C), A) and (A, (B, C), (C, B)) are duplicates.

See *3.2.7.4. Repeated expressions* for additional information on the rules for duplication.

1.10.1.18 TAG_EXTENDED*

(WARNING)

a. A tag represents an extension from the schema.

Note: Often such extensions are really spelling errors and not meant to extend the schema.

Note: Annotators are discouraged from extending the schema unless absolutely necessary. If an extension tag is needed, annotators should consider posting an **issue** explaining the tag extension so that an addition to the respective schema might be considered.

See *3.2.5 Tag extensions* for additional information on the tag extension rules.

1.10.1.19 TAG_EXTENSION_INVALID

a. A tag extension term is already in the schema.
b. A tag extension term does not comply with rules for schema nodes. **c.** A tag has extension, but an extension is not allowed.

See *3.2.5 Tag extensions* for additional information on the tag extension rules.

1.10.1.20 TAG_GROUP_ERROR

a. A tag has `tagGroup` or `topLevelTagGroup` attribute, but is not enclosed in parentheses.
b. A tag with the `topLevelTagGroup` does not appear at a HED tag group at the top level in an assembled HED annotation. **c.** Multiple tags with the `topLevelTagGroup` attribute appear in the same top-level tag group. Note: a single `Duration` and a single `Delay` may be in the same group.

See *3.2.7.2. Tag group attributes* for additional information on the rules for group errors due to schema attributes.

1.10.1.21 TAG_INVALID

a. The tag is not valid in the schema it is associated with.

See *3.2.2. Tag forms* for a discussion of tag forms and their relationship to the HED schema.

1.10.1.22 TAG_NAMESPACE_PREFIX_INVALID

a. A tag starting with *name:* does not have an associated schema.
b. A tag prefix has invalid characters.

See *3.2.6. Tag namespace prefixes* and *7. Library schema* for additional information on using multiple schemas in annotation.

1.10.1.23 TAG_NOT_UNIQUE

a. A tag with `unique` attribute appears more than once in an event-level HED string.

See *3.2.10.2. Event-level processing* for additional information on the `unique` tag.

1.10.1.24 TAG_REQUIRES_CHILD

a. A tag has the `requireChild` schema attribute but does not have a child.

See *3.2.4. Tags that take values* for an explanation of the `requireChild` attribute.

1.10.1.25 TEMPORAL_TAG_ERROR

Note: For the purpose of `Onset`/`Offset` matching, `Def` or `Def-expand` tags with different placeholder substitutions are considered to be different.

a. An `Offset`, `Onset`, `Inset`, `Duration`, or `Delay` tag does not appear in a tag group.
b. An `Offset`, `Onset`, `Inset`, `Duration`, or `Delay` tag appears in a nested tag group (not a top-level tag group).
c. An `Onset`, `Offset` or `Inset` tag is not grouped with exactly one `Def` tag or `Def-expand` group.
d. An `Onset`, `Inset`, `Duration`, or `Delay` tag group contains more than one additional tag group.
e. An `Offset` appears with one or more tags or additional tag groups.
f. An `Offset` tag appears before an `Onset` tag associated with the same definition.
g. An `Offset` tag associated with a given definition appears after a previous `Offset` tag. without the appearance of an intervening `Onset` of the same name.
h. An `Onset` or an `Inset` tag group with has tags besides the anchor `Def` tag or `Def-expand` group that are not in a tag group.
i. An `Onset` or `Offset` with a given `Def` tag or `Def-expand` group anchor appears in an event marker with the same time as with another `Onset`, `Inset`, or `Offset` that uses the same anchor.
j. An `Inset` tag is not grouped with a `Def` tag or a `Def-expand` group corresponding to an ongoing `Onset`.
k. An `Onset`, `Inset`, or `Offset` tag appears in an annotation for a non-time tabular file. **l.** A `Duration` or `Delay` tag group contains extra tags or groups, or is missing the required group. **m.** An `Offset`, `Onset`, `Inset`, `Duration`, or `Delay` tag appears with other top level tags, except Delay and Duration which can be paired.

Note: if the `Onset` tag group's definition is in expanded form, the `Def-expand` will be an additional internal tag group.

See *3.2.8.3 Onset, Offset, and Inset* for a specification of the required behavior of the `Onset`, `Offset`, and `Inset` tags.

5.3.1. Using Onset and Offset in Chapter 5 gives examples of usage and additional details.

1.10.1.26 TILDES_UNSUPPORTED

The tilde notation is not supported.

a. The **tilde syntax is no longer supported** for any version of HED.
Annotators should replace the syntax (`A ~ B ~ C`) with (`A, (B, C)`).
b. The tilde (~) is considered an invalid character in all versions of the schema.

1.10.1.27 UNITS_INVALID

a. A tag has a value with units that are invalid or not of the correct unit class for the tag.
b. A unit modifier is applied to units that are not SI units.

1.10.1.28 UNITS_MISSING*

(WARNING)

a. A tag that takes value and has a unit class does not have units.

See *3.2.4 Tags that take values* for more information.

1.10.1.29 VALUE_INVALID

a. The value substituted for a placeholder (#) is not valid.
b. A tag value is incompatible with the specified value class.
c. A tag value with no value class is assumed to be a text and contains invalid characters. **d.** The units are not separated from the value by a single blank.

See *3.2.4 Tags that take values* for more information.

1.10.1.30 VERSION_DEPRECATED*

(WARNING)

a. The HED schema version being used as been deprecated.

It is strongly recommended that a current schema version be used as these deprecated versions may not be supported in the future. Deprecated versions can be found in the **standard_schema/hedxml/deprecated** subdirectory or the corresponding subdirectory for individual library schemas in the **hed-standard/hed-schemas** GitHub repository.

Note: Support for versions of the schema less than 8.0.0 is being phased out. If you are using a deprecated version, you may need to switch to an earlier version of the HED validators.

1.10.2 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.

1.10.2.1 B.2.1. General validation errors

SCHEMA_ATTRIBUTE_INVALID

a. An attribute is used in the schema, but is not defined in the schema attribute section.
b. A schema attribute is applied to the incorrect type (e.g., an element with the unit definition does appear under an appropriate unit class).
c. A schema attribute is used in an invalid way

Attribute	Invalid Usage Location
deprecatedFrom	See *SCHEMA_DEPRECATION_ERROR*
rooted	See *SCHEMA_LIBRARY_INVALID*
takesValue	Used on a non-placeholder(#) node.
unitClass	Used on a non-placeholder(#) node.
valueClass	Used on a non-placeholder(#) node.

Notes:

- A `tag` (referred to as a node element in the schema) can have schema attributes that have the `nodeClassProperty` or the `elementProperty` or have no type property designator.

- A `unitClass` element can only have schema attributes that have the `unitClassProperty` or the `elementProperty`.

- A `unitModifier` element can only have schema attributes that have the `unitModifierProperty` or the `elementProperty`.

- A `unit` element can only have schema attributes that have the `unitProperty` or the `elementProperty`.

- A `valueClass` element can only have schema attributes that have the `valueClassProperty` or the `elementProperty`.

SCHEMA_ATTRIBUTE_VALUE_INVALID

a. A non-boolean schema attribute has an invalid value or usage as indicated by the following table.

Attribute	Invalid Attribute Value
allowedCharacter	Not a single character or one of:`letters`, `blank`, `digits`, `alphanumeric`.
conversionFactor	Not a positive numeric value.
defaultUnits	Not a valid unit in this unit class.
deprecatedFrom	See *SCHEMA_DEPRECATION_ERROR*
inLibrary	The value of an inLibrary attribute is for the wrong library.
relatedTag	Not an existing tag.
rooted	See *SCHEMA_LIBRARY_INVALID*
suggestedTag	Not an existing tag.
unitClass	Not an existing unit class.
valueClass	Not an existing value class.

SCHEMA_CHARACTER_INVALID

a. A non-placeholder schema node contains non-`name` characters. **b.** A unit class name contains non-`name` characters. **c.** A value or unit class name contains non-`name` or blank characters. **d.** The prologue or epilogue contain characters other than `text` or newline.

See *2.2 Character sets and restrictions* for definitions of the different types of characters.

Note: tag extensions may contain `nonascii` characters.

SCHEMA_DEPRECATION_ERROR

a. The value of `deprecatedFrom` is not a previously released HED schema version.
b. A deprecated tag is used as a `suggestedTag` or a `relatedTag` in a non-deprecated tag.
c. A child tag of a deprecated tag does not have the `deprecatedFrom` attribute.
d. A deprecated attribute is used on a non-deprecated element. **e.** A deprecated property is used on a non-deprecated attribute. **f.** A deprecated unit class has non-deprecated units. **g.** A tag has deprecated unit or value classes. **h.** A deprecated unit class has non-deprecated units. **i.** A unit class has a deprecated default unit

SCHEMA_DUPLICATE_NODE

a. A schema node name appears in the schema more than once.

SCHEMA_HEADER_INVALID

a. The schema header has invalid characters or format.
b. The schema header has unrecognized attributes.

SCHEMA_LIBRARY_INVALID

Library schema errors are specific to library schema. Library schema may also raise any of the other schema errors.
a. The specified library name is not alphabetic or lowercase.
b. The `withStandard` attribute is used in a header that does not also have the `library` attribute.
c. The `withStandard` attribute value does not correspond to a valid standard schema version.
d. The `rooted` attribute appears in a schema whose header does not have `unmerged="true"` as well as appropriate `library` and `withStandard` header values.
e. A node with the `rooted` attribute is not at the top level.
f. A node with the `rooted` attribute does not correspond to a node in its partnered standard schema.
g. A library schema with the `unmerged="true"` header attribute has an `inLibrary` attribute in some element.
h. A library schema with the `unmerged="true"` duplicates special section items found in its partnered standard schema.

SCHEMA_SECTION_MISSING

a. A required schema section is missing.
b. The required sections (corresponding to the prologue, schema, unit classes, unit modifiers, value classes, schema attributes, properties and epilogue) are not in the correct order and hence not detected.

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

SCHEMA_VERSION_INVALID

a. The schema version in the HED line or element is invalid.
b. A HED version specification does not have the correct syntax for the schema file format.
c. A HED schema version does not comply with semantic versioning.

1.10.2.2 B.2.3. Mediawiki format errors

WIKI_DELIMITERS_INVALID

a. Delimiters used in the wiki are invalid.
b. Schema line content after node name is not enclosed with `<nowiki></nowiki>` delimiters.
c. A line has unmatched or multiple `<nowiki></nowiki>`, [ ], or { } delimiters.
d. Attributes section of a node is malformed(e.g., hanging = character).

WIKI_LINE_START_INVALID

a. Start of body line not `'''` or *.

WIKI_SEPARATOR_INVALID

a. A malformed section separator is present
b. A duplicate section separator is present.

1.10.2.3 B.2.4. XML format errors

XML_SYNTAX_INVALID

a. XML syntax or does not comply with specified XSD.

1.10.2.4 B.2.5 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 `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 `SCHEMA_LOAD_FAILED` issue will also be reported and no additional HED validation will occur.

INDICES AND TABLES

- genindex
- search