HED Python

Release 0.4.0

HED Working Group

Apr 28, 2025

CONTENTS:

Links

- PDF docs
- Source code

Note: this is a work in progress. More information is coming.

CHAPTER

ONE

INTRODUCTION TO HED

Contents

1.1 Why HED?

Why use HED?

HED (Hierarchical Event Descriptors) is an infrastructure and a controlled vocabulary that allows researchers to annotate their experimental data, especially events, so that tools can automatically use this information in analysis.

For more information on using Hierarchical Event Descriptors (HED) visit HED examples:

1.2 Installing hedtools

Hedtools will be available soon on pypi, but in the meantime, you can install directly from the GitHub repository using the following command:

```
`code >>> pip install git+https://github.com/hed-standard/hed-python.git `
```

1.3 Finding help

Documentation

See HED resources for user documentation and tutorials.

The HED online tools provide an easy-to-use interface that requires no programming.

Issues and problems

- If you notice a bug in the python hedtools code or encounter other problems using the tools, please open an issue in the hed-python repository on github.

CHAPTER

TWO

HED TOOLS USER GUIDE

CHAPTER

THREE

HED API REFERENCE

errors	Error handling module for HED.
models	Data structures for HED tag handling.
schema	Data structures for handling the HED schema.
tools	HED remodeling, analysis and summarization tools.
validator	Validation of HED tags.

3.1 errors

Error handling module for HED.

Modules

hed.errors.error_messages	Format templates for HED schema error messages.
hed.errors.error_reporter	"
hed.errors.error_types	Error codes used in different error messages.
hed.errors.exceptions	HED exceptions and exception codes.
hed.errors.known_error_codes	Known error codes as reported in the HED specification.
hed.errors.schema_error_messages	Format templates for HED schema error messages.

3.1.1 error_messages

Format templates for HED schema error messages.

Add new errors here, or any other file imported after error_reporter.py.

Functions

def_error_bad_location(tag)	
def_error_bad_prop_in_definition(tag, def_name)	
def_error_def_tag_in_definition(tag, def_name)	
def_error_duplicate_definition(def_name)	
def_error_invalid_def_extension(tag, def_name)	
def_error_no_group_tags(def_name)	
def_error_no_takes_value(def_name, ...)	
def_error_wrong_number_groups(def_name, tag_list)	
def_error_wrong_number_tags(def_name, tag_list)	
def_error_wrong_placeholder_count(def_name, ...)	
invalid_column_ref(bad_ref)	
malformed_column_ref(column_name, index, symbol)	
nested_column_ref(column_name, ref_column)	
onset_duration_has_other_tags(tag)	
onset_duration_wrong_number_groups(tag, tag_list)	
onset_error_def_unmatched(tag)	
onset_error_inset_before_onset(tag)	
onset_error_offset_before_onset(tag)	
onset_error_same_defs_one_row(tag, def_name)	
onset_no_def_found(tag)	
onset_too_many_defs(tag, tag_list)	
onset_too_many_groups(tag, tag_list)	
onset_wrong_placeholder(tag, has_placeholder)	
onset_wrong_type_tag(tag, def_tag)	

continues on next page

Table 1 – continued from previous page

self_column_ref(self_ref)	
sidecar_error_blank_hed_string()	
sidecar_error_hed_data_type(expected_type, ...)	
sidecar_error_invalid_pound_sign_count(...)	
sidecar_error_too_many_pound_signs(...)	
sidecar_error_unknown_column(column_name)	
sidecar_hed_used()	
sidecar_na_used(column_name)	
val_error_bad_def_expand(tag, actual_def, ...)	
val_error_comma_missing(tag)	
val_error_curly_brace_unsupported_here(tag, ...)	
val_error_def_expand_unmatched(tag)	
val_error_def_expand_value_extra(tag)	
val_error_def_expand_value_missing(tag)	
val_error_def_unmatched(tag)	
val_error_def_value_extra(tag)	
val_error_def_value_missing(tag)	
val_error_duplicate_column(column_number, ...)	
val_error_duplicate_column_between_sources(...)	
val_error_duplicate_group(group)	
val_error_duplicate_reserved_tag(tag, group)	
val_error_duplicate_tag(tag)	
val_error_element_deprecated(tag)	
val_error_empty_group(tag)	
val_error_extra_column(column_name)	

continues on next page

Table 1 – continued from previous page

val_error_extra_comma(source_string, char_index)	
val_error_extra_slashes_spaces(tag, problem_tag)	
val_error_group_for_reserved_tag(group, ...)	
val_error_hed_blank_column(column_number)	
val_error_hed_placeholder_out_of_context(tag)	
val_error_invalid_char(source_string, char_index)	
val_error_invalid_extension(tag)	
val_error_invalid_parent(tag, problem_tag, ...)	
val_error_invalid_tag_character(tag, problem_tag)	
val_error_invalid_unit(tag, units)	
val_error_invalid_value_class_value(tag, ...)	
val_error_missing_column(column_name, ...)	
val_error_multiple_unique(tag_namespace)	
val_error_no_valid_tag(tag, problem_tag)	
val_error_no_value(tag[, value_class])	
val_error_onsets_unordered()	
val_error_parentheses(...)	
val_error_prefix_invalid(tag, tag_namespace)	
val_error_require_child(tag)	
val_error_sidecar_key_missing(invalid_keys, ...)	
val_error_sidecar_with_column(column_names)	
val_error_tag_extended(tag, problem_tag)	
val_error_tag_group_tag(tag)	
val_error_tags_in_group_with_reserved(tag, group)	
val_error_temporal_tag_no_time(tag)	

continues on next page

Table 1 – continued from previous page

val_error_tildes_not_supported(...)
val_error_top_level_tag(tag)
val_error_top_level_tags(tag, multiple_tags)
val_error_tsv_column_missing(invalid_keys)
val_error_unknown_namespace(tag, ...)
val_error_val_error_invalid_value_class_character(...)
val_warning_capitalization(tag)
val_warning_required_prefix_missing(...)

def_error_bad_location(*tag*)

def_error_bad_prop_in_definition(*tag*, *def_name*)

def_error_def_tag_in_definition(*tag*, *def_name*)

def_error_duplicate_definition(*def_name*)

def_error_invalid_def_extension(*tag*, *def_name*)

def_error_no_group_tags(*def_name*)

def_error_no_takes_value(*def_name*, *placeholder_tag*)

def_error_wrong_number_groups(*def_name*, *tag_list*)

def_error_wrong_number_tags(*def_name*, *tag_list*)

def_error_wrong_placeholder_count(*def_name*, *expected_count*, *tag_list*)

invalid_column_ref(*bad_ref*)

malformed_column_ref(*column_name*, *index*, *symbol*)

nested_column_ref(*column_name*, *ref_column*)

onset_duration_has_other_tags(*tag*)

onset_duration_wrong_number_groups(*tag*, *tag_list*)

onset_error_def_unmatched(*tag*)

onset_error_inset_before_onset(*tag*)

onset_error_offset_before_onset(*tag*)

onset_error_same_defs_one_row(*tag*, *def_name*)

onset_no_def_found(*tag*)

onset_too_many_defs(*tag*, *tag_list*)

onset_too_many_groups(*tag*, *tag_list*)

onset_wrong_placeholder(*tag*, *has_placeholder*)

onset_wrong_type_tag(*tag*, *def_tag*)

self_column_ref(*self_ref*)

sidecar_error_blank_hed_string()

sidecar_error_hed_data_type(*expected_type*, *given_type*)

sidecar_error_invalid_pound_sign_count(*pound_sign_count*)

sidecar_error_too_many_pound_signs(*pound_sign_count*)

sidecar_error_unknown_column(*column_name*)

sidecar_hed_used()

sidecar_na_used(*column_name*)

val_error_bad_def_expand(*tag*, *actual_def*, *found_def*)

val_error_comma_missing(*tag*)

val_error_curly_brace_unsupported_here(*tag*, *problem_tag*)

val_error_def_expand_unmatched(*tag*)

val_error_def_expand_value_extra(*tag*)

val_error_def_expand_value_missing(*tag*)

val_error_def_unmatched(*tag*)

val_error_def_value_extra(*tag*)

val_error_def_value_missing(*tag*)

val_error_duplicate_column(*column_number*, *column_name*, *list_name*)

val_error_duplicate_column_between_sources(*column_number*, *column_name*, *list_names*)

val_error_duplicate_group(*group*)

val_error_duplicate_reserved_tag(*tag*, *group*)

val_error_duplicate_tag(*tag*)

val_error_element_deprecated(*tag*)

val_error_empty_group(*tag*)

val_error_extra_column(*column_name*)

val_error_extra_comma(*source_string*, *char_index*)

val_error_extra_slashes_spaces(*tag*, *problem_tag*)

val_error_group_for_reserved_tag(*group*, *group_count*)

val_error_hed_blank_column(*column_number*)

val_error_hed_placeholder_out_of_context(*tag*)

val_error_invalid_char(*source_string*, *char_index*)

val_error_invalid_extension(*tag*)

val_error_invalid_parent(*tag*, *problem_tag*, *expected_parent_tag*)

val_error_invalid_tag_character(*tag*, *problem_tag*)

val_error_invalid_unit(*tag*, *units*)

val_error_invalid_value_class_value(*tag*, *problem_tag*, *value_class*)

val_error_missing_column(*column_name*, *column_type*)

val_error_multiple_unique(*tag_namespace*)

val_error_no_valid_tag(*tag*, *problem_tag*)

val_error_no_value(*tag*, *value_class=''*)

val_error_onsets_unordered()

val_error_parentheses(*opening_parentheses_count*, *closing_parentheses_count*)

val_error_prefix_invalid(*tag*, *tag_namespace*)

val_error_require_child(*tag*)

val_error_sidecar_key_missing(*invalid_keys*, *category_keys*)

val_error_sidecar_with_column(*column_names*)

val_error_tag_extended(*tag*, *problem_tag*)

val_error_tag_group_tag(*tag*)

val_error_tags_in_group_with_reserved(*tag*, *group*)

val_error_temporal_tag_no_time(*tag*)

val_error_tildes_not_supported(*source_string*, *char_index*)

val_error_top_level_tag(*tag*)

val_error_top_level_tags(*tag*, *multiple_tags*)

val_error_tsv_column_missing(*invalid_keys*)

val_error_unknown_namespace(*tag*, *unknown_prefix*, *known_prefixes*)

val_error_val_error_invalid_value_class_character(*tag*, *problem_tag*, *value_class*)

val_warning_capitalization(*tag*)

val_warning_required_prefix_missing(*tag_namespace*)

3.1.2 error_reporter

” Support functions for reporting validation errors.

You can scope the formatted errors with calls to push_error_context and pop_error_context.

Functions

check_for_any_errors(issues_list)	Return True if there are any errors with a severity of warning.
create_doc_link(error_code)	If error code is a known code, return a documentation url for it.
get_printable_issue_string(issues[, title, ...])	Return a string with issues list flatted into single string, one per line.
get_printable_issue_string_html(issues[, ...])	Return a string with issues list as an HTML tree.
hed_error(error_type[, default_severity, ...])	Decorator for errors in error handler or inherited classes.
hed_tag_error(error_type[, ...])	Decorator for errors in error handler or inherited classes.
iter_errors(issues)	An iterator over issues represented as flat dictionaries.
replace_tag_references(list_or_dict)	Utility function to remove any references to tags, strings, etc.
sort_issues(issues[, reverse])	Sort a list of issues by the error context values.

check_for_any_errors(*issues_list*)

Return True if there are any errors with a severity of warning.

create_doc_link(*error_code*)

If error code is a known code, return a documentation url for it.

Parameters
error_code (*str*) – A HED error code.

Returns
The URL if it’s a valid code.

Return type
url(str or None)

get_printable_issue_string(*issues*, *title=None*, *severity=None*, *skip_filename=True*, *add_link=False*)

Return a string with issues list flatted into single string, one per line.

Parameters

- **issues** (*list*) – Issues to print.
- **title** (*str*) – Optional title that will always show up first if present(even if there are no validation issues).
- **severity** (*int*) – Return only warnings >= severity.
- **skip_filename** (*bool*) – If True, don’t add the filename context to the printable string.
- **add_link** (*bool*) – Add a link at the end of message to the appropriate error if True

Returns
A string containing printable version of the issues or ‘’.

Return type
str

get_printable_issue_string_html(*issues*, *title=None*, *severity=None*, *skip_filename=True*)

Return a string with issues list as an HTML tree.

Parameters

- **issues** (*list*) – Issues to print.
- **title** (*str*) – Optional title that will always show up first if present.
- **severity** (*int*) – Return only warnings >= severity.
- **skip_filename** (*bool*) – If True, don't add the filename context to the printable string.

Returns

An HTML string containing the issues or ''.

Return type

str

hed_error(*error_type*, *default_severity=1*, *actual_code=None*)

Decorator for errors in error handler or inherited classes.

Parameters

- **error_type** (*str*) – A value from error_types or optionally another value.
- **default_severity** (*ErrorSeverity*) – The default severity for the decorated error.
- **actual_code** (*str*) – The actual error to report to the outside world.

hed_tag_error(*error_type*, *default_severity=1*, *has_sub_tag=False*, *actual_code=None*)

Decorator for errors in error handler or inherited classes.

Parameters

- **error_type** (*str*) – A value from error_types or optionally another value.
- **default_severity** (*ErrorSeverity*) – The default severity for the decorated error.
- **has_sub_tag** (*bool*) – If True, this error message also wants a sub_tag passed down. eg "This" in "This/Is/A/Tag"
- **actual_code** (*str*) – The actual error to report to the outside world.

iter_errors(*issues*)

An iterator over issues represented as flat dictionaries.

Parameters

issues (*list*) – Issues to iterator over.

Yields

dict – Represents the information in a single error.

replace_tag_references(*list_or_dict*)

Utility function to remove any references to tags, strings, etc. from any type of nested list or dict.

Use this if you want to save out issues to a file.

If you'd prefer a copy returned, use replace_tag_references(list_or_dict.copy()).

Parameters

list_or_dict (*list or dict*) – An arbitrarily nested list/dict structure

sort_issues(*issues*, *reverse=False*)

Sort a list of issues by the error context values.

Parameters

- **issues** (*list*) – A list of dictionaries representing the issues to be sorted.
- **reverse** (*bool, optional*) – If True, sorts the list in descending order. Default is False.

Returns

The sorted list of issues.

Return type

list

Classes

ErrorHandler([check_for_warnings])	Class to hold error context and having general error functions.

3.1.2.1 ErrorHandler

class ErrorHandler(*check_for_warnings=True*)

Class to hold error context and having general error functions.

Methods

ErrorHandler.__init__([check_for_warnings])	
ErrorHandler.add_context_and_filter(issues)	Filter out warnings if requested, while adding context to issues.
ErrorHandler.filter_issues_by_severity(...)	Gather all issues matching or below a given severity.
ErrorHandler.format_error(error_type, *args)	Format an error based on the parameters, which vary based on what type of error this is.
ErrorHandler.format_error_from_context(...)	Format an error based on the error type.
ErrorHandler.format_error_with_context(...)	
ErrorHandler.pop_error_context()	Remove the last scope from the error context.
ErrorHandler.push_error_context(...)	Push a new error context to narrow down error scope.
ErrorHandler.reset_error_context()	Reset all error context information to defaults.
ErrorHandler.val_error_unknown(**kwargs)	Default error handler if no error of this type was registered.

Attributes

ErrorHandler.**__init__**(*check_for_warnings=True*)

ErrorHandler.**add_context_and_filter**(*issues*)

Filter out warnings if requested, while adding context to issues.

issues(list):
list: A list containing a single dictionary representing a single error.

static ErrorHandler.**filter_issues_by_severity**(*issues_list*, *severity*)

Gather all issues matching or below a given severity.

Parameters

- **issues_list** (*list*) – A list of dictionaries containing the full issue list.
- **severity** (*int*) – The level of issues to keep.

Returns
A list of dictionaries containing the issue list after filtering by severity.

Return type
list

static ErrorHandler.**format_error**(*error_type*, **args*, *actual_error=None*, ***kwargs*)

Format an error based on the parameters, which vary based on what type of error this is.

Parameters

- **error_type** (*str*) – The type of error for this. Registered with @hed_error or @hed_tag_error.
- **args** (*args*) – Any remaining non keyword args after those required by the error type.
- **actual_error** (*str or None*) – Code to actually add to report out.
- **kwargs** (*kwargs*) – The other keyword args to pass down to the error handling func.

Returns
A list containing a single dictionary representing a single error.

Return type
list

Notes

The actual error is useful for errors that are shared like invalid character.

static ErrorHandler.**format_error_from_context**(*error_type*, *error_context*, **args*, *actual_error=None*, ***kwargs*)

Format an error based on the error type.

Parameters

- **error_type** (*str*) – The type of error. Registered with @hed_error or @hed_tag_error.
- **error_context** (*list*) – Contains the error context to use for this error.
- **args** (*args*) – Any remaining non keyword args.

- **actual_error** (*str or None*) – Error code to actually add to report out.
- **kwargs** (*kwargs*) – Keyword parameters to pass down to the error handling func.

Returns
A list containing a single dictionary.

Return type
list

Notes

- Generally the error_context is returned from _add_context_to_errors.
- The actual_error is useful for errors that are shared like invalid character.
- This can't filter out warnings like the other ones.

ErrorHandler.**format_error_with_context**(**args*, ***kwargs*)

ErrorHandler.**pop_error_context**()
Remove the last scope from the error context.

Notes

Modifies the error context of this reporter.

ErrorHandler.**push_error_context**(*context_type*, *context*)
Push a new error context to narrow down error scope.

Parameters

- **context_type** (*ErrorContext*) – A value from ErrorContext representing the type of scope.
- **context** (*str, int, or HedString*) – The main value for the context_type.

Notes

The context depends on the context_type. For ErrorContext.FILE_NAME this would be the actual filename.

ErrorHandler.**reset_error_context**()
Reset all error context information to defaults.

Notes

This function is mainly for testing and should not be needed with proper usage.

ErrorHandler.**val_error_unknown**(***kwargs*)
Default error handler if no error of this type was registered.

Parameters

- **args** (*args*) – List of non-keyword parameters (varies).
- **kwargs** (*kwargs*) – Keyword parameters (varies)

Returns
The error message.

Return type
str

3.1.3 error_types

Error codes used in different error messages.

Classes

ColumnErrors()	
DefinitionErrors()	
ErrorContext()	Context this error took place in, each error potentially having multiple contexts.
ErrorSeverity()	Severity codes for errors
SchemaAttributeErrors()	
SchemaErrors()	
SchemaWarnings()	
SidecarErrors()	
TemporalErrors()	
ValidationErrors()	

3.1.3.1 ColumnErrors

class ColumnErrors

Methods

ColumnErrors.__init__()	

Attributes

ColumnErrors.INVALID_COLUMN_REF
ColumnErrors.MALFORMED_COLUMN_REF
ColumnErrors.NESTED_COLUMN_REF
ColumnErrors.SELF_COLUMN_REF

ColumnErrors.**__init__**()

ColumnErrors.**INVALID_COLUMN_REF** = **'INVALID_COLUMN_REF'**

ColumnErrors.**MALFORMED_COLUMN_REF** = **'MALFORMED_COLUMN_REF'**

ColumnErrors.**NESTED_COLUMN_REF** = **'NESTED_COLUMN_REF'**

ColumnErrors.**SELF_COLUMN_REF** = **'SELF_COLUMN_REF'**

3.1.3.2 DefinitionErrors

class DefinitionErrors

Methods

DefinitionErrors.__init__()

Attributes

DefinitionErrors.BAD_DEFINITION_LOCATION
DefinitionErrors.BAD_PROP_IN_DEFINITION
DefinitionErrors.DEF_TAG_IN_DEFINITION
DefinitionErrors.DUPLICATE_DEFINITION
DefinitionErrors.INVALID_DEFINITION_EXTENSION
DefinitionErrors.NO_DEFINITION_CONTENTS
DefinitionErrors.PLACEHOLDER_NO_TAKES_VALUE
DefinitionErrors.WRONG_NUMBER_GROUPS
DefinitionErrors.WRONG_NUMBER_PLACEHOLDER_TAGS
DefinitionErrors.WRONG_NUMBER_TAGS

DefinitionErrors.**__init__**()

DefinitionErrors.**BAD_DEFINITION_LOCATION** = 'BAD_DEFINITION_LOCATION'

DefinitionErrors.**BAD_PROP_IN_DEFINITION** = 'BAD_PROP_IN_DEFINITION'

DefinitionErrors.**DEF_TAG_IN_DEFINITION** = 'DEF_TAG_IN_DEFINITION'

DefinitionErrors.**DUPLICATE_DEFINITION** = 'duplicateDefinition'

DefinitionErrors.**INVALID_DEFINITION_EXTENSION** = 'invalidDefExtension'

DefinitionErrors.**NO_DEFINITION_CONTENTS** = 'NO_DEFINITION_CONTENTS'

DefinitionErrors.**PLACEHOLDER_NO_TAKES_VALUE** = 'PLACEHOLDER_NO_TAKES_VALUE'

DefinitionErrors.**WRONG_NUMBER_GROUPS** = 'WRONG_NUMBER_GROUPS'

DefinitionErrors.**WRONG_NUMBER_PLACEHOLDER_TAGS** = 'wrongNumberPlaceholderTags'

DefinitionErrors.**WRONG_NUMBER_TAGS** = 'WRONG_NUMBER_TAGS'

3.1.3.3 ErrorContext

class ErrorContext

Context this error took place in, each error potentially having multiple contexts.

Methods

ErrorContext.__init__()

Attributes

ErrorContext.COLUMN
ErrorContext.CUSTOM_TITLE
ErrorContext.FILE_NAME
ErrorContext.HED_STRING
ErrorContext.LINE
ErrorContext.ROW
ErrorContext.SCHEMA_ATTRIBUTE
ErrorContext.SCHEMA_SECTION
ErrorContext.SCHEMA_TAG
ErrorContext.SIDECAR_COLUMN_NAME
ErrorContext.SIDECAR_KEY_NAME

ErrorContext.**__init__**()

ErrorContext.**COLUMN** = 'ec_column'

ErrorContext.**CUSTOM_TITLE** = 'ec_title'

ErrorContext.**FILE_NAME** = 'ec_filename'

ErrorContext.**HED_STRING** = 'ec_HedString'

ErrorContext.**LINE** = 'ec_line'

ErrorContext.**ROW** = 'ec_row'

ErrorContext.**SCHEMA_ATTRIBUTE** = 'ec_attribute'

ErrorContext.**SCHEMA_SECTION** = 'ec_section'

ErrorContext.**SCHEMA_TAG** = 'ec_schema_tag'

ErrorContext.**SIDECAR_COLUMN_NAME** = 'ec_sidecarColumnName'

ErrorContext.**SIDECAR_KEY_NAME** = 'ec_sidecarKeyName'

3.1.3.4 ErrorSeverity

class ErrorSeverity

Severity codes for errors

Methods

ErrorSeverity.__init__()

Attributes

ErrorSeverity.ERROR

ErrorSeverity.WARNING

ErrorSeverity.**__init__**()

ErrorSeverity.**ERROR = 1**

ErrorSeverity.**WARNING = 10**

3.1.3.5 SchemaAttributeErrors

class SchemaAttributeErrors

Methods

SchemaAttributeErrors.__init__()

Attributes

SchemaAttributeErrors. *SCHEMA_ALLOWED_CHARACTERS_INVALID*
SchemaAttributeErrors. *SCHEMA_ATTRIBUTE_INVALID*
SchemaAttributeErrors. *SCHEMA_ATTRIBUTE_NUMERIC_INVALID*
SchemaAttributeErrors. *SCHEMA_ATTRIBUTE_VALUE_DEPRECATED*
SchemaAttributeErrors. *SCHEMA_ATTRIBUTE_VALUE_INVALID*
SchemaAttributeErrors. *SCHEMA_CHILD_OF_DEPRECATED*
SchemaAttributeErrors. *SCHEMA_CONVERSION_FACTOR_NOT_POSITIVE*
SchemaAttributeErrors. *SCHEMA_DEFAULT_UNITS_DEPRECATED*
SchemaAttributeErrors. *SCHEMA_DEFAULT_UNITS_INVALID*
SchemaAttributeErrors. *SCHEMA_DEPRECATED_INVALID*
SchemaAttributeErrors. *SCHEMA_DEPRECATION_ERROR*
SchemaAttributeErrors. *SCHEMA_GENERIC_ATTRIBUTE_VALUE_INVALID*
SchemaAttributeErrors. *SCHEMA_HED_ID_INVALID*
SchemaAttributeErrors. *SCHEMA_IN_LIBRARY_INVALID*

SchemaAttributeErrors.**__init__**()

SchemaAttributeErrors.**SCHEMA_ALLOWED_CHARACTERS_INVALID** = **'SCHEMA_ALLOWED_CHARACTERS_INVALID'**

SchemaAttributeErrors.**SCHEMA_ATTRIBUTE_INVALID** = **'SCHEMA_ATTRIBUTE_INVALID'**

SchemaAttributeErrors.**SCHEMA_ATTRIBUTE_NUMERIC_INVALID** = **'SCHEMA_ATTRIBUTE_NUMERIC_INVALID'**

SchemaAttributeErrors.**SCHEMA_ATTRIBUTE_VALUE_DEPRECATED** = **'SCHEMA_ATTRIBUTE_VALUE_DEPRECATED'**

SchemaAttributeErrors.**SCHEMA_ATTRIBUTE_VALUE_INVALID** = **'SCHEMA_ATTRIBUTE_VALUE_INVALID'**

SchemaAttributeErrors.**SCHEMA_CHILD_OF_DEPRECATED** = **'SCHEMA_CHILD_OF_DEPRECATED'**

SchemaAttributeErrors.**SCHEMA_CONVERSION_FACTOR_NOT_POSITIVE** = **'SCHEMA_CONVERSION_FACTOR_NOT_POSITIVE'**

SchemaAttributeErrors.**SCHEMA_DEFAULT_UNITS_DEPRECATED** = **'SCHEMA_DEFAULT_UNITS_DEPRECATED'**

SchemaAttributeErrors.**SCHEMA_DEFAULT_UNITS_INVALID** = **'SCHEMA_DEFAULT_UNITS_INVALID'**

SchemaAttributeErrors.**SCHEMA_DEPRECATED_INVALID = 'SCHEMA_DEPRECATED_INVALID'**

SchemaAttributeErrors.**SCHEMA_DEPRECATION_ERROR = 'SCHEMA_DEPRECATION_ERROR'**

SchemaAttributeErrors.**SCHEMA_GENERIC_ATTRIBUTE_VALUE_INVALID = 'SCHEMA_GENERIC_ATTRIBUTE_VALUE_INVALID'**

SchemaAttributeErrors.**SCHEMA_HED_ID_INVALID = 'SCHEMA_HED_ID_INVALID'**

SchemaAttributeErrors.**SCHEMA_IN_LIBRARY_INVALID = 'SCHEMA_IN_LIBRARY_INVALID'**

3.1.3.6 SchemaErrors

class SchemaErrors

Methods

SchemaErrors.__init__()

Attributes

SchemaErrors.SCHEMA_DUPLICATE_FROM_LIBRARY

SchemaErrors.SCHEMA_DUPLICATE_NODE

SchemaErrors.SCHEMA_INVALID_CHILD

SchemaErrors.SCHEMA_INVALID_SIBLING

SchemaErrors.**__init__**()

SchemaErrors.**SCHEMA_DUPLICATE_FROM_LIBRARY = 'SCHEMA_LIBRARY_INVALID'**

SchemaErrors.**SCHEMA_DUPLICATE_NODE = 'SCHEMA_DUPLICATE_NODE'**

SchemaErrors.**SCHEMA_INVALID_CHILD = 'SCHEMA_INVALID_CHILD'**

SchemaErrors.**SCHEMA_INVALID_SIBLING = 'SCHEMA_INVALID_SIBLING'**

3.1.3.7 SchemaWarnings

class SchemaWarnings

Methods

SchemaWarnings.__init__()

Attributes

SchemaWarnings.SCHEMA_CHARACTER_INVALID

SchemaWarnings.SCHEMA_INVALID_CAPITALIZATION

SchemaWarnings.SCHEMA_INVALID_CHARACTERS_IN_DESC

SchemaWarnings.SCHEMA_INVALID_CHARACTERS_IN_TAG

SchemaWarnings.SCHEMA_NON_PLACEHOLDER_HAS_CLASS

SchemaWarnings.SCHEMA_PRERELEASE_VERSION_USED

SchemaWarnings.SCHEMA_PROLOGUE_CHARACTER_INVALID

SchemaWarnings.**__init__**()

SchemaWarnings.**SCHEMA_CHARACTER_INVALID = 'SCHEMA_CHARACTER_INVALID'**

SchemaWarnings.**SCHEMA_INVALID_CAPITALIZATION = 'invalidCaps'**

SchemaWarnings.**SCHEMA_INVALID_CHARACTERS_IN_DESC = 'SCHEMA_INVALID_CHARACTERS_IN_DESC'**

SchemaWarnings.**SCHEMA_INVALID_CHARACTERS_IN_TAG = 'SCHEMA_INVALID_CHARACTERS_IN_TAG'**

SchemaWarnings.**SCHEMA_NON_PLACEHOLDER_HAS_CLASS = 'SCHEMA_NON_PLACEHOLDER_HAS_CLASS'**

SchemaWarnings.**SCHEMA_PRERELEASE_VERSION_USED = 'SCHEMA_PRERELEASE_VERSION_USED'**

SchemaWarnings.**SCHEMA_PROLOGUE_CHARACTER_INVALID = 'SCHEMA_PROLOGUE_CHARACTER_INVALID'**

3.1.3.8 SidecarErrors

class SidecarErrors

Methods

SidecarErrors.__init__()

Attributes

SidecarErrors.BLANK_HED_STRING
SidecarErrors.INVALID_POUND_SIGNS_CATEGORY
SidecarErrors.INVALID_POUND_SIGNS_VALUE
SidecarErrors.SIDECAR_BRACES_INVALID
SidecarErrors.SIDECAR_HED_USED
SidecarErrors.SIDECAR_NA_USED
SidecarErrors.UNKNOWN_COLUMN_TYPE
SidecarErrors.WRONG_HED_DATA_TYPE

SidecarErrors.**__init__**()

SidecarErrors.**BLANK_HED_STRING** = **'blankValueString'**

SidecarErrors.**INVALID_POUND_SIGNS_CATEGORY** = **'tooManyPoundSigns'**

SidecarErrors.**INVALID_POUND_SIGNS_VALUE** = **'invalidNumberPoundSigns'**

SidecarErrors.**SIDECAR_BRACES_INVALID** = **'SIDECAR_BRACES_INVALID'**

SidecarErrors.**SIDECAR_HED_USED** = **'SIDECAR_HED_USED'**

SidecarErrors.**SIDECAR_NA_USED** = **'SIDECAR_NA_USED'**

SidecarErrors.**UNKNOWN_COLUMN_TYPE** = **'sidecarUnknownColumn'**

SidecarErrors.**WRONG_HED_DATA_TYPE** = **'wrongHedDataType'**

3.1.3.9 TemporalErrors

class TemporalErrors

Methods

TemporalErrors.__init__()

Attributes

TemporalErrors.DURATION_HAS_OTHER_TAGS
TemporalErrors.DURATION_WRONG_NUMBER_GROUPS
TemporalErrors.INSET_BEFORE_ONSET
TemporalErrors.OFFSET_BEFORE_ONSET
TemporalErrors.ONSET_DEF_UNMATCHED
TemporalErrors.ONSET_NO_DEF_TAG_FOUND
TemporalErrors.ONSET_PLACEHOLDER_WRONG
TemporalErrors.ONSET_SAME_DEFS_ONE_ROW
TemporalErrors.ONSET_TAG_OUTSIDE_OF_GROUP
TemporalErrors.ONSET_TOO_MANY_DEFS
TemporalErrors.ONSET_WRONG_NUMBER_GROUPS
TemporalErrors.TEMPORAL_TAG_NO_TIME

TemporalErrors.**__init__**()

TemporalErrors.**DURATION_HAS_OTHER_TAGS** = 'DURATION_HAS_OTHER_TAGS'

TemporalErrors.**DURATION_WRONG_NUMBER_GROUPS** = 'DURATION_WRONG_NUMBER_GROUPS'

TemporalErrors.**INSET_BEFORE_ONSET** = 'INSET_BEFORE_ONSET'

TemporalErrors.**OFFSET_BEFORE_ONSET** = 'OFFSET_BEFORE_ONSET'

TemporalErrors.**ONSET_DEF_UNMATCHED** = 'ONSET_DEF_UNMATCHED'

TemporalErrors.**ONSET_NO_DEF_TAG_FOUND** = 'ONSET_NO_DEF_TAG_FOUND'

TemporalErrors.**ONSET_PLACEHOLDER_WRONG** = 'ONSET_PLACEHOLDER_WRONG'

TemporalErrors.**ONSET_SAME_DEFS_ONE_ROW** = 'ONSET_SAME_DEFS_ONE_ROW'

TemporalErrors.**ONSET_TAG_OUTSIDE_OF_GROUP** = 'ONSET_TAG_OUTSIDE_OF_GROUP'

TemporalErrors.**ONSET_TOO_MANY_DEFS** = 'ONSET_TOO_MANY_DEFS'

TemporalErrors.**ONSET_WRONG_NUMBER_GROUPS = 'ONSET_WRONG_NUMBER_GROUPS'**

TemporalErrors.**TEMPORAL_TAG_NO_TIME = 'TEMPORAL_TAG_NO_TIME'**

3.1.3.10 ValidationErrors

class ValidationErrors

Methods

ValidationErrors.__init__()

Attributes

ValidationErrors.CHARACTER_INVALID

ValidationErrors.COMMA_MISSING

ValidationErrors.CURLY_BRACE_UNSUPPORTED_HERE

ValidationErrors.DEFINITION_INVALID

ValidationErrors.DEF_EXPAND_INVALID

ValidationErrors.DEF_INVALID

ValidationErrors.DUPLICATE_COLUMN_BETWEEN_SOURCES

ValidationErrors.DUPLICATE_COLUMN_IN_LIST

ValidationErrors.ELEMENT_DEPRECATED

ValidationErrors.HED_BLANK_COLUMN

ValidationErrors.HED_COLUMN_MISSING

ValidationErrors.HED_DEF_EXPAND_INVALID

ValidationErrors.HED_DEF_EXPAND_UNMATCHED

ValidationErrors.HED_DEF_EXPAND_VALUE_EXTRA

ValidationErrors.HED_DEF_EXPAND_VALUE_MISSING

ValidationErrors.HED_DEF_UNMATCHED

continues on next page

Table 2 – continued from previous page

ValidationErrors.HED_DEF_VALUE_EXTRA
ValidationErrors.HED_DEF_VALUE_MISSING
ValidationErrors.HED_GROUP_EMPTY
ValidationErrors.HED_LIBRARY_UNMATCHED
ValidationErrors.HED_MISSING_REQUIRED_COLUMN
ValidationErrors.HED_MULTIPLE_TOP_TAGS
ValidationErrors.HED_PLACEHOLDER_OUT_OF_CONTEXT
ValidationErrors.HED_RESERVED_TAG_GROUP_ERROR
ValidationErrors.HED_RESERVED_TAG_REPEATED
ValidationErrors.HED_TAGS_NOT_ALLOWED
ValidationErrors.HED_TAG_GROUP_TAG
ValidationErrors.HED_TAG_REPEATED
ValidationErrors.HED_TAG_REPEATED_GROUP
ValidationErrors.HED_TOP_LEVEL_TAG
ValidationErrors.HED_UNKNOWN_COLUMN
ValidationErrors.INVALID_PARENT_NODE
ValidationErrors.INVALID_TAG_CHARACTER
ValidationErrors.INVALID_VALUE_CLASS_CHARACTER
ValidationErrors.INVALID_VALUE_CLASS_VALUE
ValidationErrors.NODE_NAME_EMPTY
ValidationErrors.NO_VALID_TAG_FOUND
ValidationErrors.ONSETS_UNORDERED
ValidationErrors.PARENTHESES_MISMATCH
ValidationErrors.PLACEHOLDER_INVALID
ValidationErrors.REQUIRED_TAG_MISSING

continues on next page

Table 2 – continued from previous page

ValidationErrors.SIDECAR_AND_OTHER_COLUMNS
ValidationErrors.SIDECAR_INVALID
ValidationErrors.SIDECAR_KEY_MISSING
ValidationErrors.STYLE_WARNING
ValidationErrors.TAG_EMPTY
ValidationErrors.TAG_EXPRESSION_REPEATED
ValidationErrors.TAG_EXTENDED
ValidationErrors.TAG_EXTENSION_INVALID
ValidationErrors.TAG_GROUP_ERROR
ValidationErrors.TAG_INVALID
ValidationErrors.TAG_NAMESPACE_PREFIX_INVALID
ValidationErrors.TAG_NOT_UNIQUE
ValidationErrors.TAG_REQUIRES_CHILD
ValidationErrors.TEMPORAL_TAG_ERROR
ValidationErrors.TILDES_UNSUPPORTED
ValidationErrors.TSV_COLUMN_MISSING
ValidationErrors.UNITS_INVALID
ValidationErrors.VALUE_INVALID
ValidationErrors.VERSION_DEPRECATED

ValidationErrors.**__init__**()

ValidationErrors.**CHARACTER_INVALID** = 'CHARACTER_INVALID'

ValidationErrors.**COMMA_MISSING** = 'COMMA_MISSING'

ValidationErrors.**CURLY_BRACE_UNSUPPORTED_HERE** = 'CURLY_BRACE_UNSUPPORTED_HERE'

ValidationErrors.**DEFINITION_INVALID** = 'DEFINITION_INVALID'

ValidationErrors.**DEF_EXPAND_INVALID** = 'DEF_EXPAND_INVALID'

ValidationErrors.**DEF_INVALID** = 'DEF_INVALID'

ValidationErrors.DUPLICATE_COLUMN_BETWEEN_SOURCES = 'DUPLICATE_COLUMN_BETWEEN_SOURCES'

ValidationErrors.DUPLICATE_COLUMN_IN_LIST = 'DUPLICATE_COLUMN_IN_LIST'

ValidationErrors.ELEMENT_DEPRECATED = 'ELEMENT_DEPRECATED'

ValidationErrors.HED_BLANK_COLUMN = 'HED_BLANK_COLUMN'

ValidationErrors.HED_COLUMN_MISSING = 'HED_COLUMN_MISSING'

ValidationErrors.HED_DEF_EXPAND_INVALID = 'HED_DEF_EXPAND_INVALID'

ValidationErrors.HED_DEF_EXPAND_UNMATCHED = 'HED_DEF_EXPAND_UNMATCHED'

ValidationErrors.HED_DEF_EXPAND_VALUE_EXTRA = 'HED_DEF_EXPAND_VALUE_EXTRA'

ValidationErrors.HED_DEF_EXPAND_VALUE_MISSING = 'HED_DEF_EXPAND_VALUE_MISSING'

ValidationErrors.HED_DEF_UNMATCHED = 'HED_DEF_UNMATCHED'

ValidationErrors.HED_DEF_VALUE_EXTRA = 'HED_DEF_VALUE_EXTRA'

ValidationErrors.HED_DEF_VALUE_MISSING = 'HED_DEF_VALUE_MISSING'

ValidationErrors.HED_GROUP_EMPTY = 'HED_GROUP_EMPTY'

ValidationErrors.HED_LIBRARY_UNMATCHED = 'HED_LIBRARY_UNMATCHED'

ValidationErrors.HED_MISSING_REQUIRED_COLUMN = 'HED_MISSING_REQUIRED_COLUMN'

ValidationErrors.HED_MULTIPLE_TOP_TAGS = 'HED_MULTIPLE_TOP_TAGS'

ValidationErrors.HED_PLACEHOLDER_OUT_OF_CONTEXT = 'HED_PLACEHOLDER_OUT_OF_CONTEXT'

ValidationErrors.HED_RESERVED_TAG_GROUP_ERROR = 'HED_RESERVED_TAG_GROUP_ERROR'

ValidationErrors.HED_RESERVED_TAG_REPEATED = 'HED_RESERVED_TAG_REPEATED'

ValidationErrors.HED_TAGS_NOT_ALLOWED = 'HED_TAGS_NOT_ALLOWED'

ValidationErrors.HED_TAG_GROUP_TAG = 'HED_TAG_GROUP_TAG'

ValidationErrors.HED_TAG_REPEATED = 'HED_TAG_REPEATED'

ValidationErrors.HED_TAG_REPEATED_GROUP = 'HED_TAG_REPEATED_GROUP'

ValidationErrors.HED_TOP_LEVEL_TAG = 'HED_TOP_LEVEL_TAG'

ValidationErrors.HED_UNKNOWN_COLUMN = 'HED_UNKNOWN_COLUMN'

ValidationErrors.INVALID_PARENT_NODE = 'invalidParent'

ValidationErrors.INVALID_TAG_CHARACTER = 'invalidTagCharacter'

ValidationErrors.INVALID_VALUE_CLASS_CHARACTER = 'INVALID_VALUE_CLASS_CHARACTER'

ValidationErrors.INVALID_VALUE_CLASS_VALUE = 'INVALID_VALUE_CLASS_VALUE'

ValidationErrors.NODE_NAME_EMPTY = 'NODE_NAME_EMPTY'

ValidationErrors.NO_VALID_TAG_FOUND = 'invalidTag'

ValidationErrors.**ONSETS_UNORDERED = 'ONSETS_UNORDERED'**

ValidationErrors.**PARENTHESES_MISMATCH = 'PARENTHESES_MISMATCH'**

ValidationErrors.**PLACEHOLDER_INVALID = 'PLACEHOLDER_INVALID'**

ValidationErrors.**REQUIRED_TAG_MISSING = 'REQUIRED_TAG_MISSING'**

ValidationErrors.**SIDECAR_AND_OTHER_COLUMNS = 'SIDECAR_AND_OTHER_COLUMNS'**

ValidationErrors.**SIDECAR_INVALID = 'SIDECAR_INVALID'**

ValidationErrors.**SIDECAR_KEY_MISSING = 'SIDECAR_KEY_MISSING'**

ValidationErrors.**STYLE_WARNING = 'STYLE_WARNING'**

ValidationErrors.**TAG_EMPTY = 'TAG_EMPTY'**

ValidationErrors.**TAG_EXPRESSION_REPEATED = 'TAG_EXPRESSION_REPEATED'**

ValidationErrors.**TAG_EXTENDED = 'TAG_EXTENDED'**

ValidationErrors.**TAG_EXTENSION_INVALID = 'TAG_EXTENSION_INVALID'**

ValidationErrors.**TAG_GROUP_ERROR = 'TAG_GROUP_ERROR'**

ValidationErrors.**TAG_INVALID = 'TAG_INVALID'**

ValidationErrors.**TAG_NAMESPACE_PREFIX_INVALID = 'TAG_NAMESPACE_PREFIX_INVALID'**

ValidationErrors.**TAG_NOT_UNIQUE = 'TAG_NOT_UNIQUE'**

ValidationErrors.**TAG_REQUIRES_CHILD = 'TAG_REQUIRES_CHILD'**

ValidationErrors.**TEMPORAL_TAG_ERROR = 'TEMPORAL_TAG_ERROR'**

ValidationErrors.**TILDES_UNSUPPORTED = 'TILDES_UNSUPPORTED'**

ValidationErrors.**TSV_COLUMN_MISSING = 'TSV_COLUMN_MISSING'**

ValidationErrors.**UNITS_INVALID = 'UNITS_INVALID'**

ValidationErrors.**VALUE_INVALID = 'VALUE_INVALID'**

ValidationErrors.**VERSION_DEPRECATED = 'VERSION_DEPRECATED'**

3.1.4 exceptions

HED exceptions and exception codes.

Classes

HedExceptions()	HED exception codes.

3.1.4.1 HedExceptions

class HedExceptions

HED exception codes.

Methods

HedExceptions.__init__()

Attributes

HedExceptions.BAD_COLUMN_NAMES
HedExceptions.BAD_HED_LIBRARY_NAME
HedExceptions.BAD_PARAMETERS
HedExceptions.BAD_WITH_STANDARD
HedExceptions.BAD_WITH_STANDARD_MULTIPLE_VALUES
HedExceptions.CANNOT_PARSE_JSON
HedExceptions.CANNOT_PARSE_RDF
HedExceptions.CANNOT_PARSE_XML
HedExceptions.FILE_NOT_FOUND
HedExceptions.GENERIC_ERROR
HedExceptions.HED_SCHEMA_NODE_NAME_INVALID
HedExceptions.INVALID_DATAFRAME
HedExceptions.INVALID_EXTENSION
HedExceptions.INVALID_FILE_FORMAT
HedExceptions.INVALID_HED_FORMAT

continues on next page

Table 3 – continued from previous page

HedExceptions.INVALID_LIBRARY_PREFIX
HedExceptions.IN_LIBRARY_IN_UNMERGED
HedExceptions.ROOTED_TAG_DOES_NOT_EXIST
HedExceptions.ROOTED_TAG_HAS_PARENT
HedExceptions.ROOTED_TAG_INVALID
HedExceptions.SCHEMA_DUPLICATE_LIBRARY
HedExceptions.SCHEMA_DUPLICATE_NAMES
HedExceptions.SCHEMA_DUPLICATE_PREFIX
HedExceptions.SCHEMA_HEADER_INVALID
HedExceptions.SCHEMA_HEADER_MISSING
HedExceptions.SCHEMA_INVALID
HedExceptions.SCHEMA_LIBRARY_INVALID
HedExceptions.SCHEMA_LOAD_FAILED
HedExceptions.SCHEMA_SECTION_MISSING
HedExceptions.SCHEMA_TAG_TSV_BAD_PARENT
HedExceptions.SCHEMA_UNKNOWN_HEADER_ATTRIBUTE
HedExceptions.SCHEMA_VERSION_INVALID
HedExceptions.URL_ERROR
HedExceptions.WIKI_DELIMITERS_INVALID
HedExceptions.WIKI_LINE_INVALID
HedExceptions.WIKI_LINE_START_INVALID
HedExceptions.WIKI_SEPARATOR_INVALID

HedExceptions.**__init__**()

HedExceptions.**BAD_COLUMN_NAMES** = **'BAD_COLUMN_NAMES'**

HedExceptions.**BAD_HED_LIBRARY_NAME** = **'SCHEMA_LIBRARY_INVALID'**

HedExceptions.**BAD_PARAMETERS** = **'badParameters'**

HedExceptions.BAD_WITH_STANDARD = 'SCHEMA_LIBRARY_INVALID'

HedExceptions.BAD_WITH_STANDARD_MULTIPLE_VALUES = 'SCHEMA_LOAD_FAILED'

HedExceptions.CANNOT_PARSE_JSON = 'cannotParseJson'

HedExceptions.CANNOT_PARSE_RDF = 'CANNOT_PARSE_RDF'

HedExceptions.CANNOT_PARSE_XML = 'cannotParseXML'

HedExceptions.FILE_NOT_FOUND = 'fileNotFound'

HedExceptions.GENERIC_ERROR = 'GENERIC_ERROR'

HedExceptions.HED_SCHEMA_NODE_NAME_INVALID = 'HED_SCHEMA_NODE_NAME_INVALID'

HedExceptions.INVALID_DATAFRAME = 'INVALID_DATAFRAME'

HedExceptions.INVALID_EXTENSION = 'invalidExtension'

HedExceptions.INVALID_FILE_FORMAT = 'INVALID_FILE_FORMAT'

HedExceptions.INVALID_HED_FORMAT = 'INVALID_HED_FORMAT'

HedExceptions.INVALID_LIBRARY_PREFIX = 'SCHEMA_LIBRARY_INVALID'

HedExceptions.IN_LIBRARY_IN_UNMERGED = 'SCHEMA_LIBRARY_INVALID'

HedExceptions.ROOTED_TAG_DOES_NOT_EXIST = 'SCHEMA_LIBRARY_INVALID'

HedExceptions.ROOTED_TAG_HAS_PARENT = 'SCHEMA_LIBRARY_INVALID'

HedExceptions.ROOTED_TAG_INVALID = 'SCHEMA_LIBRARY_INVALID'

HedExceptions.SCHEMA_DUPLICATE_LIBRARY = 'SCHEMA_LIBRARY_INVALID'

HedExceptions.SCHEMA_DUPLICATE_NAMES = 'SCHEMA_DUPLICATE_NAMES'

HedExceptions.SCHEMA_DUPLICATE_PREFIX = 'SCHEMA_LOAD_FAILED'

HedExceptions.SCHEMA_HEADER_INVALID = 'SCHEMA_HEADER_INVALID'

HedExceptions.SCHEMA_HEADER_MISSING = 'SCHEMA_HEADER_INVALID'

HedExceptions.SCHEMA_INVALID = 'SCHEMA_INVALID'

HedExceptions.SCHEMA_LIBRARY_INVALID = 'SCHEMA_LIBRARY_INVALID'

HedExceptions.SCHEMA_LOAD_FAILED = 'SCHEMA_LOAD_FAILED'

HedExceptions.SCHEMA_SECTION_MISSING = 'SCHEMA_SECTION_MISSING'

HedExceptions.SCHEMA_TAG_TSV_BAD_PARENT = 'SCHEMA_TAG_TSV_BAD_PARENT'

HedExceptions.SCHEMA_UNKNOWN_HEADER_ATTRIBUTE = 'SCHEMA_HEADER_INVALID'

HedExceptions.SCHEMA_VERSION_INVALID = 'SCHEMA_VERSION_INVALID'

HedExceptions.URL_ERROR = 'URL_ERROR'

HedExceptions.WIKI_DELIMITERS_INVALID = 'WIKI_DELIMITERS_INVALID'

HedExceptions.**WIKI_LINE_INVALID = 'WIKI_LINE_INVALID'**

HedExceptions.**WIKI_LINE_START_INVALID = 'WIKI_LINE_START_INVALID'**

HedExceptions.**WIKI_SEPARATOR_INVALID = 'invalidSectionSeparator'**

Exceptions

HedFileError(code, message, filename[, issues])	Exception raised when a file cannot be parsed due to being malformed, file IO, etc.

3.1.4.2 hed.errors.exceptions.HedFileError

exception HedFileError(*code*, *message*, *filename*, *issues=None*)

Exception raised when a file cannot be parsed due to being malformed, file IO, etc.

3.1.5 known_error_codes

Known error codes as reported in the HED specification.

3.1.6 schema_error_messages

Format templates for HED schema error messages.

Functions

schema_error_GENERIC_ATTRIBUTE_VALUE_INVALID(...)	
schema_error_SCHEMA_ALLOWED_CHARACTERS_INVALID(...)	
schema_error_SCHEMA_ATTRIBUTE_NUMERIC_INVALID(...)	
schema_error_SCHEMA_ATTRIBUTE_VALUE_DEPRECATED(...)	
schema_error_SCHEMA_CHILD_OF_DEPRECATED(...)	
schema_error_SCHEMA_CONVERSION_FACTOR_NOT_POSITIVE(...)	
schema_error_SCHEMA_DEFAULT_UNITS_DEPRECATED(...)	
schema_error_SCHEMA_DEFAULT_UNITS_INVALID(...)	
schema_error_SCHEMA_DEPRECATED_INVALID(...)	
schema_error_SCHEMA_HED_ID_INVALID(tag, new_id)	
schema_error_SCHEMA_INVALID_CHILD(tag, ...)	
schema_error_SCHEMA_INVALID_SIBLING(tag, ...)	
schema_error_SCHEMA_IN_LIBRARY_INVALID(tag, ...)	
schema_error_SCHEMA_PRERELEASE_VERSION_USED(...)	
schema_error_hed_duplicate_from_library(tag, ...)	
schema_error_hed_duplicate_node(tag, ...)	
schema_error_invalid_character_prologue(...)	
schema_error_unknown_attribute(...)	
schema_warning_SCHEMA_INVALID_CAPITALIZATION(...)	
schema_warning_invalid_chars_desc(...)	
schema_warning_invalid_chars_tag(tag_name, ...)	
schema_warning_non_placeholder_class(...)	

schema_error_GENERIC_ATTRIBUTE_VALUE_INVALID(*tag*, *invalid_value*, *attribute_name*)

schema_error_SCHEMA_ALLOWED_CHARACTERS_INVALID(*tag*, *invalid_character*)

schema_error_SCHEMA_ATTRIBUTE_NUMERIC_INVALID(*tag*, *invalid_value*, *attribute_name*)

schema_error_SCHEMA_ATTRIBUTE_VALUE_DEPRECATED(*tag*, *deprecated_suggestion*, *attribute_name*)

schema_error_SCHEMA_CHILD_OF_DEPRECATED(*deprecated_tag*, *non_deprecated_child*)

schema_error_SCHEMA_CONVERSION_FACTOR_NOT_POSITIVE(*tag*, *conversion_factor*)

schema_error_SCHEMA_DEFAULT_UNITS_DEPRECATED(*unit_class*, *bad_unit*)

schema_error_SCHEMA_DEFAULT_UNITS_INVALID(*tag*, *bad_unit*, *valid_units*)

schema_error_SCHEMA_DEPRECATED_INVALID(*tag_name*, *invalid_deprecated_version*)

schema_error_SCHEMA_HED_ID_INVALID(*tag*, *new_id*, *old_id=None*, *valid_min=None*, *valid_max=None*)

schema_error_SCHEMA_INVALID_CHILD(*tag*, *child_tag_list*)

schema_error_SCHEMA_INVALID_SIBLING(*tag*, *sibling_tag_list*)

schema_error_SCHEMA_IN_LIBRARY_INVALID(*tag*, *bad_library*)

schema_error_SCHEMA_PRERELEASE_VERSION_USED(*current_version*, *known_versions*)

schema_error_hed_duplicate_from_library(*tag*, *duplicate_tag_list*, *section*)

schema_error_hed_duplicate_node(*tag*, *duplicate_tag_list*, *section*)

schema_error_invalid_character_prologue(*char_index*, *source_string*, *section_name*)

schema_error_unknown_attribute(*attribute_name*, *source_tag*)

schema_warning_SCHEMA_INVALID_CAPITALIZATION(*tag_name*, *problem_char*, *char_index*)

schema_warning_invalid_chars_desc(*desc_string*, *tag_name*, *problem_char*, *char_index*)

schema_warning_invalid_chars_tag(*tag_name*, *problem_char*, *char_index*)

schema_warning_non_placeholder_class(*tag_name*, *invalid_attribute_name*)

3.2 models

Data structures for HED tag handling.

Modules

hed.models.base_input	Superclass representing a basic columnar file.
hed.models.basic_search	Utilities to support HED searches based on strings.
hed.models.basic_search_util	Utilities to support HED searches based on strings.
hed.models.column_mapper	Mapping of a base input file columns into HED tags.
hed.models.column_metadata	Column type for a column in a ColumnMapper.
hed.models.def_expand_gather	Classes to resolve ambiguities, gather, expand definitions.
hed.models.definition_dict	Definition handler class.
hed.models.definition_entry	A single definition.
hed.models.df_util	Utilities for assembly and conversion of HED strings to different forms.
hed.models.hed_group	A single parenthesized HED string.
hed.models.hed_string	A HED string with its schema and definitions.
hed.models.hed_tag	A single HED tag.
hed.models.model_constants	Defined constants for definitions, def labels, and expanded labels.
hed.models.query_expressions	Classes representing parsed query expressions.
hed.models.query_handler	Holder for and manipulation of search results.
hed.models.query_service	Functions to get and use HED queries.
hed.models.query_util	Classes representing HED search results and tokens.
hed.models.sidecar	Contents of a JSON file or merged JSON files.
hed.models.spreadsheet_input	A spreadsheet of HED tags.
hed.models.string_util	Utilities for manipulating HedString objects.
hed.models.tabular_input	A BIDS tabular file with sidecar.
hed.models.timeseries_input	A BIDS time series tabular file.

3.2.1 base_input

Superclass representing a basic columnar file.

Classes

`BaseInput`(file[, file_type, worksheet_name, ...])	Superclass representing a basic columnar file.

3.2.1.1 BaseInput

class BaseInput(*file*, *file_type=None*, *worksheet_name=None*, *has_column_names=True*, *mapper=None*, *name=None*, *allow_blank_names=True*)

Superclass representing a basic columnar file.

Methods

BaseInput.__init__(file[, file_type, ...])	Constructor for the BaseInput class.
BaseInput.assemble([mapper, skip_curly_braces])	Assembles the HED strings.
BaseInput.column_metadata()	Return the metadata for each column.
BaseInput.combine_dataframe(dataframe)	Combine all columns in the given dataframe into a single HED string series,
BaseInput.convert_to_form(hed_schema, tag_form)	Convert all tags in underlying dataframe to the specified form.
BaseInput.convert_to_long(hed_schema)	Convert all tags in underlying dataframe to long form.
BaseInput.convert_to_short(hed_schema)	Convert all tags in underlying dataframe to short form.
BaseInput.expand_defs(hed_schema, def_dict)	Shrinks any def-expand found in the underlying dataframe.
BaseInput.get_column_refs()	Return a list of column refs for this file.
BaseInput.get_def_dict(hed_schema[, ...])	Return the definition dict for this file.
BaseInput.get_worksheet([worksheet_name])	Get the requested worksheet.
BaseInput.reset_mapper(new_mapper)	Set mapper to a different view of the file.
BaseInput.set_cell(row_number, ...[, tag_form])	Replace the specified cell with transformed text.
BaseInput.shrink_defs(hed_schema)	Shrinks any def-expand found in the underlying dataframe.
BaseInput.to_csv([file])	Write to file or return as a string.
BaseInput.to_excel(file)	Output to an Excel file.
BaseInput.validate(hed_schema[, ...])	Creates a SpreadsheetValidator and returns all issues with this file.

Attributes

BaseInput.EXCEL_EXTENSION	
BaseInput.TEXT_EXTENSION	
BaseInput.columns	Returns a list of the column names.
BaseInput.dataframe	The underlying dataframe.
BaseInput.dataframe_a	Return the assembled dataframe Probably a placeholder name.
BaseInput.has_column_names	True if dataframe has column names.
BaseInput.loaded_workbook	The underlying loaded workbooks.
BaseInput.name	Name of the data.
BaseInput.needs_sorting	Return True if this both has an onset column, and it needs sorting.
BaseInput.onsets	Return the onset column if it exists.
BaseInput.series_a	Return the assembled dataframe as a series.
BaseInput.series_filtered	Return the assembled dataframe as a series, with rows that have the same onset combined.
BaseInput.worksheet_name	The worksheet name.

BaseInput.**__init__**(*file*, *file_type=None*, *worksheet_name=None*, *has_column_names=True*, *mapper=None*, *name=None*, *allow_blank_names=True*)

Constructor for the BaseInput class.

Parameters

- **file** (*str or file-like or pd.Dataframe*) – An xlsx/tsv file to open.
- **file_type** (*str or None*) – ".xlsx" (Excel), ".tsv" or ".txt" (tab-separated text). Derived from file if file is a filename. Ignored if pandas dataframe.
- **worksheet_name** (*str or None*) – Name of Excel workbook worksheet name to use. (Not applicable to tsv files.)
- **has_column_names** (*bool*) – True if file has column names. This value is ignored if you pass in a pandas dataframe.
- **mapper** (*ColumnMapper or None*) – Indicates which columns have HED tags. See SpreadsheetInput or TabularInput for examples of how to use built-in a ColumnMapper.
- **name** (*str or None*) – Optional field for how this file will report errors.
- **allow_blank_names** (*bool*) – If True, column names can be blank

Raises

HedFileError –

- file is blank.
- An invalid dataframe was passed with size 0.
- An invalid extension was provided.
- A duplicate or empty column name appears.
- Cannot open the indicated file.
- The specified worksheet name does not exist.
- If the sidecar file or tabular file had invalid format and could not be read.

BaseInput.**assemble**(*mapper=None*, *skip_curly_braces=False*)

Assembles the HED strings.

Parameters

- **mapper** (*ColumnMapper or None*) – Generally pass none here unless you want special behavior.
- **skip_curly_braces** (*bool*) – If True, don't plug in curly brace values into columns.

Returns

The assembled dataframe.

Return type

Dataframe

BaseInput.**column_metadata**()

Return the metadata for each column.

Returns

Number/ColumnMeta pairs.

Return type

dict

static BaseInput.**combine_dataframe**(*dataframe*)

Combine all columns in the given dataframe into a single HED string series,

skipping empty columns and columns with empty strings.

Parameters
dataframe (*Dataframe*) – The dataframe to combine

Returns
The assembled series.

Return type
Series

BaseInput.**convert_to_form**(*hed_schema*, *tag_form*)

Convert all tags in underlying dataframe to the specified form.

Parameters

- **hed_schema** (*HedSchema*) – The schema to use to convert tags.
- **tag_form** (*str*) – HedTag property to convert tags to. Most cases should use convert_to_short or convert_to_long below.

BaseInput.**convert_to_long**(*hed_schema*)

Convert all tags in underlying dataframe to long form.

Parameters
hed_schema (*HedSchema or None*) – The schema to use to convert tags.

BaseInput.**convert_to_short**(*hed_schema*)

Convert all tags in underlying dataframe to short form.

Parameters
hed_schema (*HedSchema*) – The schema to use to convert tags.

BaseInput.**expand_defs**(*hed_schema*, *def_dict*)

Shrinks any def-expand found in the underlying dataframe.

Parameters

- **hed_schema** (*HedSchema or None*) – The schema to use to identify defs.
- **def_dict** (*DefinitionDict*) – The definitions to expand.

BaseInput.**get_column_refs**()

Return a list of column refs for this file.

Default implementation returns none.

Returns
A list of unique column refs found.

Return type
column_refs(list)

BaseInput.**get_def_dict**(*hed_schema*, *extra_def_dicts=None*)

Return the definition dict for this file.

Note: Baseclass implementation returns just extra_def_dicts.

Parameters

- **hed_schema** (*HedSchema*) – Identifies tags to find definitions(if needed).
- **extra_def_dicts** (*list, DefinitionDict, or None*) – Extra dicts to add to the list.

Returns
A single definition dict representing all the data(and extra def dicts).

Return type
DefinitionDict

BaseInput.**get_worksheet**(*worksheet_name=None*)

Get the requested worksheet.

Parameters
worksheet_name (*str or None*) – The name of the requested worksheet by name or the first one if None.

Returns
The workbook request.

Return type
openpyxl.workbook.Workbook

Notes

If None, returns the first worksheet.

Raises
KeyError –

- The specified worksheet name does not exist.

BaseInput.**reset_mapper**(*new_mapper*)

Set mapper to a different view of the file.

Parameters
new_mapper (*ColumnMapper*) – A column mapper to be associated with this base input.

BaseInput.**set_cell**(*row_number*, *column_number*, *new_string_obj*, *tag_form='short_tag'*)

Replace the specified cell with transformed text.

Parameters

- **row_number** (*int*) – The row number of the spreadsheet to set.
- **column_number** (*int*) – The column number of the spreadsheet to set.
- **new_string_obj** (*HedString*) – Object with text to put in the given cell.
- **tag_form** (*str*) – Version of the tags (short_tag, long_tag, base_tag, etc.)

Notes

Any attribute of a HedTag that returns a string is a valid value of tag_form.

Raises

- **ValueError** –
  - There is not a loaded dataframe.
- **KeyError** –
  - The indicated row/column does not exist.
- **AttributeError** –

- The indicated tag_form is not an attribute of HedTag.

BaseInput.**shrink_defs**(*hed_schema*)

Shrinks any def-expand found in the underlying dataframe.

Parameters
hed_schema (*HedSchema or None*) – The schema to use to identify defs.

BaseInput.**to_csv**(*file=None*)

Write to file or return as a string.

Parameters
file (*str, file-like, or None*) – Location to save this file. If None, return as string.

Returns
None if file is given or the contents as a str if file is None.

Return type
None or str

Raises
OSError –

- Cannot open the indicated file.

BaseInput.**to_excel**(*file*)

Output to an Excel file.

Parameters
file (*str or file-like*) – Location to save this base input.

Raises

- **ValueError** –
  - If empty file object was passed.
- **OSError** –
  - Cannot open the indicated file.

BaseInput.**validate**(*hed_schema*, *extra_def_dicts=None*, *name=None*, *error_handler=None*)

Creates a SpreadsheetValidator and returns all issues with this file.

Parameters

- **hed_schema** (*HedSchema*) – The schema to use for validation.
- **extra_def_dicts** (*list of DefDict or DefDict*) – All definitions to use for validation.
- **name** (*str*) – The name to report errors from this file as.
- **error_handler** (*ErrorHandler*) – Error context to use. Creates a new one if None.

Returns
A list of issues for a HED string.

Return type
issues (list of dict)

BaseInput.**EXCEL_EXTENSION = ['.xlsx']**

BaseInput.**TEXT_EXTENSION = ['.tsv', '.txt']**

BaseInput.**columns**

Returns a list of the column names.

Empty if no column names.

Returns
The column names.

Return type
columns(list)

BaseInput.**dataframe**

The underlying dataframe.

BaseInput.**dataframe_a**

Return the assembled dataframe Probably a placeholder name.

Returns
the assembled dataframe

Return type
Dataframe

BaseInput.**has_column_names**

True if dataframe has column names.

BaseInput.**loaded_workbook**

The underlying loaded workbooks.

BaseInput.**name**

Name of the data.

BaseInput.**needs_sorting**

Return True if this both has an onset column, and it needs sorting.

BaseInput.**onsets**

Return the onset column if it exists.

BaseInput.**series_a**

Return the assembled dataframe as a series.

Returns
the assembled dataframe with columns merged.

Return type
Series

BaseInput.**series_filtered**

Return the assembled dataframe as a series, with rows that have the same onset combined.

Returns
the assembled dataframe with columns merged, and the rows filtered together.

Return type
Series or None

BaseInput.**worksheet_name**

The worksheet name.

3.2.2 basic_search

Utilities to support HED searches based on strings.

Functions

check_parentheses(text)	Check for balanced parentheses in the given text and returns the unbalanced ones.
construct_delimiter_map(text, words)	Based on an input search query and list of words, return the parenthetical delimiters between them.
find_matching(series, search_string[, regex])	Find lines in the series that match the search string and returns a mask.
find_words(search_string)	Extract words in the search string based on their prefixes.
reverse_and_flip_parentheses(s)	Reverse a string and flips the parentheses.
verify_search_delimiters(text, ...)	Verify that the text contains specific words with expected delimiters between them.

check_parentheses(*text*)

Check for balanced parentheses in the given text and returns the unbalanced ones.

Parameters
text (*str*) – The text to be checked for balanced parentheses.

Returns
A string containing the unbalanced parentheses in their original order.

Return type
str

Notes

- The function only considers the characters '(' and ')' for balancing.
- Balanced pairs of parentheses are removed, leaving behind only the unbalanced ones.

construct_delimiter_map(*text*, *words*)

Based on an input search query and list of words, return the parenthetical delimiters between them.

Parameters

- **text** (*str*) – The search query.
- **words** (*list*) – A list of words we want to map between from the query.

Returns
The two-way delimiter map.

Return type
dict

find_matching(*series*, *search_string*, *regex=False*)

Find lines in the series that match the search string and returns a mask.

Syntax Rules:

- '@': Prefixing a term in the search string means the term must appear anywhere within a line.

- '~': Prefixing a term in the search string means the term must NOT appear within a line.
- **Parentheses: Elements within parentheses must appear in the line with the same level of nesting.**

  e.g.: Search string: "(A), (B)" will match "(A), (B, C)", but not "(A, B)", since they don't start in the same group.
- "LongFormTag*": A * will match any remaining word(anything but a comma or parenthesis)
- An individual term can be arbitrary regex, but it is limited to single continuous words.

Notes

- **Specific words only care about their level relative to other specific words, not overall.**
  e.g. "(A, B)" will find: "A, B", "(A, B)", (A, (C), B)", or ((A, B))"
- If you have no grouping or anywhere words in the search, it assumes all terms are anywhere words.
- The format of the series should match the format of the search string, whether it's in short or long form.
- To enable support for matching parent tags, ensure that both the series and search string are in long form.

Parameters

- **series** (*pd.Series*) – A Pandas Series object containing the lines to be searched.
- **search_string** (*str*) – The string to search for in each line of the series.
- **regex** (*bool*) – By default, translate any * wildcard characters to .*? regex. If True, do no translation and pass the words as is. Due to how it's setup, you must not include the following characters: (),

Returns

A Boolean mask Series of the same length as the input series.
The mask has *True* for lines that match the search string and *False* otherwise.

Return type
mask (pd.Series)

find_words(*search_string*)

Extract words in the search string based on their prefixes.

Parameters
search_string (*str*) – The search query string to parse. Words can be prefixed with '@' or '~'.

Returns

A list containing three lists:

- Words prefixed with '@'
- Words prefixed with '~'
- Words with no prefix

Return type
list

reverse_and_flip_parentheses(*s*)

Reverse a string and flips the parentheses.

Parameters
s (*str*) – The string to be reversed and have its parentheses flipped.

Returns
The reversed string with flipped parentheses.

Return type
str

Notes

- The function takes into account only the '(' and ')' characters for flipping.

verify_search_delimiters(*text*, *specific_words*, *delimiter_map*)

Verify that the text contains specific words with expected delimiters between them.

Parameters

- **text** (*str*) – The text to search in.
- **specific_words** (*list of str*) – Words that must appear relative to other words in the text.
- **delimiter_map** (*dict*) – A dictionary specifying expected delimiters between pairs of specific words.

Returns
True if all conditions are met, otherwise False.

Return type
bool

3.2.3 basic_search_util

Utilities to support HED searches based on strings.

Functions

convert_query(search_query, schema)	Converts the given basic search query into a hed_string

convert_query(*search_query*, *schema*)

Converts the given basic search query into a hed_string

Parameters

- **search_query** (*str*) – The basic search query to convert.
- **schema** (*HedSchema*) – The schema to use to convert tags

Returns
The converted search query, in long form.

Return type
long_query(str)

3.2.4 column_mapper

Mapping of a base input file columns into HED tags.

Classes

ColumnMapper([sidecar, tag_columns, ...])	Mapping of a base input file columns into HED tags.

3.2.4.1 ColumnMapper

class ColumnMapper(*sidecar=None*, *tag_columns=None*, *column_prefix_dictionary=None*, *optional_tag_columns=None*, *warn_on_missing_column=False*)

Mapping of a base input file columns into HED tags.

Notes

- All column numbers are 0 based.

Methods

ColumnMapper.__init__([sidecar, ...])	Constructor for ColumnMapper.
ColumnMapper.check_for_blank_names(...)	Validate there are no blank column names.
ColumnMapper.check_for_mapping_issues([...])	Find all issues given the current column_map, tag_columns, etc.
ColumnMapper.get_column_mapping_issues()	Get all the issues with finalizing column mapping(duplicate columns, missing required, etc.).
ColumnMapper.get_def_dict(hed_schema[, ...])	Return def dicts from every column description.
ColumnMapper.get_tag_columns()	Return the column numbers or names that are mapped to be HedTags.
ColumnMapper.get_transformers()	Return the transformers to use on a dataframe.
ColumnMapper.set_column_map([new_column_map])	Set the column number to name mapping.
ColumnMapper.set_column_prefix_dictionary(...)	Set the column prefix dictionary.
ColumnMapper.set_tag_columns([tag_columns, ...])	Set tag columns and optional tag columns.

Attributes

ColumnMapper.column_prefix_dictionary	Return the column_prefix_dictionary with numbers turned into names where possible.
ColumnMapper.sidecar_column_data	Pass through to get the sidecar ColumnMetadata.
ColumnMapper.tag_columns	Return the known tag and optional tag columns with numbers as names when possible.

ColumnMapper.**__init__**(*sidecar=None*, *tag_columns=None*, *column_prefix_dictionary=None*, *optional_tag_columns=None*, *warn_on_missing_column=False*)

Constructor for ColumnMapper.

Parameters

- **sidecar** (*Sidecar*) – A sidecar to gather column data from.
- **tag_columns** – (list): A list of ints or strings containing the columns that contain the HED tags. Sidecar column definitions will take precedent if there is a conflict with tag_columns.
- **column_prefix_dictionary** (*dict*) – Dictionary with keys that are column numbers/names and values are HED tag prefixes to prepend to the tags in that column before processing.
- **optional_tag_columns** (*list*) – A list of ints or strings containing the columns that contain the HED tags. If the column is otherwise unspecified, convert this column type to HED-Tags.
- **warn_on_missing_column** (*bool*) – If True, issue mapping warnings on column names that are missing from the sidecar.

Notes

- All column numbers are 0 based.
- **The column_prefix_dictionary may be deprecated/renamed in the future.**
  - These are no longer prefixes, but rather converted to value columns: {"key": "Description", 1: "Label/"} will turn into value columns as {"key": "Description/#", 1: "Label/#"} It will be a validation issue if column 1 is called "key" in the above example. This means it no longer accepts anything but the value portion only in the columns.

`static ColumnMapper.check_for_blank_names(column_map, allow_blank_names)`

Validate there are no blank column names.

Parameters

- **column_map** (*iterable*) – A list of column names.
- **allow_blank_names** (*bool*) – Only find issues if True.

Returns
A list of dicts, one per issue.

Return type
issues(list)

`ColumnMapper.check_for_mapping_issues(allow_blank_names=False)`

Find all issues given the current column_map, tag_columns, etc.

Parameters
allow_blank_names (*bool*) – Only flag blank names if False.

Returns
All issues found as a list of dicts.

Return type
issue_list(list of dict)

`ColumnMapper.get_column_mapping_issues()`

Get all the issues with finalizing column mapping(duplicate columns, missing required, etc.).

Notes

- This is deprecated and now a wrapper for "check_for_mapping_issues()".

Returns
A list dictionaries of all issues found from mapping column names to numbers.

Return type
list

ColumnMapper.**get_def_dict**(*hed_schema*, *extra_def_dicts=None*)

Return def dicts from every column description.

Parameters

- **hed_schema** (*Schema*) – A HED schema object to use for extracting definitions.
- **extra_def_dicts** (*list, DefinitionDict, or None*) – Extra dicts to add to the list.

Returns
A single definition dict representing all the data(and extra def dicts).

Return type
DefinitionDict

ColumnMapper.**get_tag_columns**()

Return the column numbers or names that are mapped to be HedTags.

Note: This is NOT the tag_columns or optional_tag_columns parameter, though they set it.

Returns

A list of column numbers or names that are ColumnType.HedTags.
0-based if integer-based, otherwise column name.

Return type
column_identifiers(list)

ColumnMapper.**get_transformers**()

Return the transformers to use on a dataframe.

Returns
dict({str or int: func}): The functions to use to transform each column. need_categorical(list of int): A list of columns to treat as categorical.

Return type
tuple(dict, list)

ColumnMapper.**set_column_map**(*new_column_map=None*)

Set the column number to name mapping.

Parameters
new_column_map (*list or dict*) – Either an ordered list of the column names or column_number:column name. dictionary. In both cases, column numbers start at 0.

Returns
List of issues. Each issue is a dictionary.

Return type
list

ColumnMapper.**set_column_prefix_dictionary**(*column_prefix_dictionary*, *finalize_mapping=True*)

Set the column prefix dictionary.

ColumnMapper.**set_tag_columns**(*tag_columns=None*, *optional_tag_columns=None*, *finalize_mapping=True*)

Set tag columns and optional tag columns.

Parameters

- **tag_columns** (*list*) – A list of ints or strings containing the columns that contain the HED tags. If None, clears existing tag_columns
- **optional_tag_columns** (*list*) – A list of ints or strings containing the columns that contain the HED tags, but not an error if missing. If None, clears existing tag_columns
- **finalize_mapping** (*bool*) – Re-generate the internal mapping if True, otherwise no effect until finalize.

ColumnMapper.**column_prefix_dictionary**

Return the column_prefix_dictionary with numbers turned into names where possible.

Returns
A column_prefix_dictionary with column labels as keys.

Return type
column_prefix_dictionary(list of str or int)

ColumnMapper.**sidecar_column_data**

Pass through to get the sidecar ColumnMetadata.

Returns
ColumnMetadata}): The column metadata defined by this sidecar.

Return type
dict({str

ColumnMapper.**tag_columns**

Return the known tag and optional tag columns with numbers as names when possible.

Returns
A list of all tag and optional tag columns as labels.

Return type
tag_columns(list of str or int)

3.2.5 column_metadata

Column type for a column in a ColumnMapper.

Classes

ColumnMetadata([column_type, name, source])	Column in a ColumnMapper.
ColumnType(value)	The overall column_type of a column in column mapper, e.g.

3.2.5.1 ColumnMetadata

class ColumnMetadata(*column_type=None*, *name=None*, *source=None*)

Column in a ColumnMapper.

Methods

ColumnMetadata.__init__([column_type, name, ...])	A single column entry in the column mapper.
ColumnMetadata.expected_pound_sign_count(...)	Return how many pound signs a column string should have.
ColumnMetadata.get_hed_strings()	Return the HED strings for this entry as a series.
ColumnMetadata.set_hed_strings(new_strings)	Set the HED strings for this entry.

Attributes

ColumnMetadata.hed_dict	The HED strings for any given entry.
ColumnMetadata.source_dict	The raw dict for this entry(if it exists).

ColumnMetadata.**__init__**(*column_type=None*, *name=None*, *source=None*)

A single column entry in the column mapper.

Parameters

- **column_type** (*ColumnType or None*) – How to treat this column when reading data.
- **name** (*str, int, or None*) – The column_name or column number identifying this column. If name is a string, you'll need to use a column map to set the number later.
- **source** (*dict or str or None*) – Either the entire loaded json sidecar or a single HED string.

static ColumnMetadata.**expected_pound_sign_count**(*column_type*)

Return how many pound signs a column string should have.

Parameters
column_type (*ColumnType*) – The type of the column.

Returns
expected_count(int): The expected count. 0 or 1. error_type(str): The type of the error we should issue.

Return type
tuple

ColumnMetadata.**get_hed_strings**()

Return the HED strings for this entry as a series.

Returns
The HED strings for this series.(potentially empty).

Return type
hed_strings(pd.Series)

ColumnMetadata.**set_hed_strings**(*new_strings*)

Set the HED strings for this entry.

Parameters
new_strings (*pd.Series, dict, or str*) – The HED strings to set. This should generally be the return value from get_hed_strings.

Returns
The HED strings for this series.(potentially empty).

Return type
hed_strings(pd.Series)

ColumnMetadata.**hed_dict**

The HED strings for any given entry.

Returns
A string or dict of strings for this column.

Return type
dict or str

ColumnMetadata.**source_dict**

The raw dict for this entry(if it exists).

Returns
A string or dict of strings for this column.

Return type
dict or str

3.2.5.2 ColumnType

class ColumnType(*value*)

The overall column_type of a column in column mapper, e.g. treat it as HED tags.

Mostly internal to column mapper related code

Methods

Attributes

ColumnType.Unknown

ColumnType.Ignore

ColumnType.Categorical

ColumnType.Value

ColumnType.HEDTags

ColumnType.**Unknown = None**

ColumnType.**Ignore = 'ignore'**

ColumnType.**Categorical = 'categorical'**

ColumnType.**Value = 'value'**

ColumnType.**HEDTags = 'hed_tags'**

3.2.6 def_expand_gather

Classes to resolve ambiguities, gather, expand definitions.

Classes

AmbiguousDef()	Determine whether expanded definitions are consistent.
DefExpandGatherer(hed_schema[, known_defs, ...])	Gather definitions from a series of def-expands, including possibly ambiguous ones.

3.2.6.1 AmbiguousDef

class AmbiguousDef

Determine whether expanded definitions are consistent.

Methods

AmbiguousDef.__init__()	
AmbiguousDef.add_def(def_tag, def_expand_group)	Adds a definition to this ambiguous definition.
AmbiguousDef.get_definition_string()	
AmbiguousDef.resolve_definition()	Try to resolve the definition based on the information available.

Attributes

AmbiguousDef.**__init__**()

AmbiguousDef.**add_def**(*def_tag*, *def_expand_group*)

Adds a definition to this ambiguous definition.

Parameters

- **def_tag** (*HedTag*) – The Def-expand tag representing this definition.
- **def_expand_group** (*HedGroup*) – The Definition group including the tag and contents.

Raises

ValueError – if this group could not match any of the other possible matches.

AmbiguousDef.**get_definition_string**()

AmbiguousDef.**resolve_definition**()

Try to resolve the definition based on the information available.

Returns: boolean - True if successfully resolved and False if it can't be resolved from information available.

Raises: ValueError - If the actual_contents conflict.

If the definition has already been resolved, this rechecks based on the information.

3.2.6.2 DefExpandGatherer

class DefExpandGatherer(*hed_schema*, *known_defs=None*, *ambiguous_defs=None*, *errors=None*)

Gather definitions from a series of def-expands, including possibly ambiguous ones.

Notes: The def-dict contains the known definitions. After validation, it also contains resolved definitions. The errors contain the definition contents that are known to be in error. The ambiguous_defs contain the definitions that cannot be resolved based on the data.

Methods

DefExpandGatherer.__init__(hed_schema[, ...])	Initialize the DefExpandGatherer class.
DefExpandGatherer.process_def_expands(...[, ...])	Process the HED strings containing def-expand tags.

Attributes

DefExpandGatherer.**__init__**(*hed_schema*, *known_defs=None*, *ambiguous_defs=None*, *errors=None*)

Initialize the DefExpandGatherer class.

Parameters

- **hed_schema** (*HedSchema*) – The HED schema to be used for processing.
- **known_defs** (*str or list or DefinitionDict*) – A dictionary of known definitions.
- **ambiguous_defs** (*dict, optional*) – A dictionary of ambiguous def-expand definitions.

DefExpandGatherer.**process_def_expands**(*hed_strings*, *known_defs=None*)

Process the HED strings containing def-expand tags.

Parameters

- **hed_strings** (*pd.Series or list*) – A Pandas Series or list of HED strings to be processed.
- **known_defs** (*dict, optional*) – A dictionary of known definitions to be added.

Returns

A tuple containing the DefinitionDict, ambiguous definitions, and errors.

Return type
tuple

3.2.7 definition_dict

Definition handler class.

Classes

DefinitionDict([def_dicts, hed_schema])	Gathers definitions from a single source.

3.2.7.1 DefinitionDict

class DefinitionDict(*def_dicts=None*, *hed_schema=None*)

Gathers definitions from a single source.

Methods

DefinitionDict.__init__([def_dicts, hed_schema])	Definitions to be considered a single source.
DefinitionDict.add_definitions(def_dicts[, ...])	Add definitions from dict(s) or strings(s) to this dict.
DefinitionDict.check_for_definitions(...[, ...])	Check string for definition tags, adding them to self.
DefinitionDict.get(def_name)	Get the definition entry for the definition name.
DefinitionDict.get_as_strings(def_dict)	Convert the entries to strings of the contents
DefinitionDict.get_definition_entry(def_tag)	Get the entry for a given def tag.
DefinitionDict.items()	Return the dictionary of definitions.

Attributes

DefinitionDict.issues	Return issues about duplicate definitions.

DefinitionDict.**__init__**(*def_dicts=None*, *hed_schema=None*)

Definitions to be considered a single source.

Parameters

- **def_dicts** (*str or list or DefinitionDict*) – DefDict or list of DefDicts/strings or a single string whose definitions should be added.
- **hed_schema** (*HedSchema or None*) – Required if passing strings or lists of strings, unused otherwise.

Raises

TypeError –

- Bad type passed as def_dicts.

DefinitionDict.**add_definitions**(*def_dicts*, *hed_schema=None*)

Add definitions from dict(s) or strings(s) to this dict.

Parameters

- **def_dicts** (*list, DefinitionDict, dict, or str*) – DefinitionDict or list of DefinitionDicts/strings/dicts whose definitions should be added.
- **hed_schema** (*HedSchema or None*) – Required if passing strings or lists of strings, unused otherwise.

Note - dict form expects DefinitionEntries in the same form as a DefinitionDict

Note - str or list of strings will parse the strings using the hed_schema. Note - You can mix and match types, eg [DefinitionDict, str, list of str] would be valid input.

Raises

TypeError –

- Bad type passed as def_dicts.

DefinitionDict.**check_for_definitions**(*hed_string_obj*, *error_handler=None*)

Check string for definition tags, adding them to self.

Parameters

- **hed_string_obj** (*HedString*) – A single HED string to gather definitions from.
- **error_handler** (*ErrorHandler or None*) – Error context used to identify where definitions are found.

Returns

List of issues encountered in checking for definitions. Each issue is a dictionary.

Return type

list

DefinitionDict.**get**(*def_name*)

Get the definition entry for the definition name.

Not case-sensitive

Parameters

def_name (*str*) – Name of the definition to retrieve.

Returns

Definition entry for the requested definition.

Return type

DefinitionEntry

static DefinitionDict.**get_as_strings**(*def_dict*)

Convert the entries to strings of the contents

Parameters

def_dict (*DefinitionDict or dict*) – A dict of definitions

Returns

definition name and contents

Return type

dict(str)

DefinitionDict.**get_definition_entry**(*def_tag*)

Get the entry for a given def tag.

Does not validate at all.

Parameters
def_tag (*HedTag*) – Source HED tag that may be a Def or Def-expand tag.

Returns
The definition entry if it exists

Return type
def_entry(DefinitionEntry or None)

DefinitionDict.**items**()

Return the dictionary of definitions.

Alias for .defs.items()

Returns
DefinitionEntry}): A list of definitions.

Return type
def_entries({str

DefinitionDict.**issues**

Return issues about duplicate definitions.

3.2.8 definition_entry

A single definition.

Classes

DefinitionEntry(name, contents, takes_value, ...)	A single definition.

3.2.8.1 DefinitionEntry

class DefinitionEntry(*name*, *contents*, *takes_value*, *source_context*)

A single definition.

Methods

DefinitionEntry.__init__(name, contents, ...)	Initialize info for a single definition.
DefinitionEntry.get_definition(replace_tag)	Return a copy of the definition with the tag expanded and the placeholder plugged in.

Attributes

DefinitionEntry.__init__(*name*, *contents*, *takes_value*, *source_context*)

Initialize info for a single definition.

Parameters

- **name** (*str*) – The label portion of this name (not including Definition/).
- **contents** (*HedGroup*) – The contents of this definition (which could be None).
- **takes_value** (*bool*) – If True, expects ONE tag to have a single # sign in it.
- **source_context** (*list, None*) – List (stack) of dictionaries giving context for reporting errors.

DefinitionEntry.get_definition(*replace_tag*, *placeholder_value=None*, *return_copy_of_tag=False*)

Return a copy of the definition with the tag expanded and the placeholder plugged in.

Returns None if placeholder_value passed when it doesn't take value, or vice versa.

Parameters

- **replace_tag** (*HedTag*) – The def HED tag to replace with an expanded version.
- **placeholder_value** (*str or None*) – If present and required, will replace any pound signs in the definition contents.
- **return_copy_of_tag** (*bool*) – Set to True for validation.

Returns
The contents of this definition(including the def tag itself).

Return type
HedGroup

Raises
ValueError –

- Something internally went wrong with finding the placeholder tag. This should not be possible.

3.2.9 df_util

Utilities for assembly and conversion of HED strings to different forms.

Functions

convert_to_form(df, hed_schema, tag_form[, ...])	Convert all tags in underlying dataframe to the specified form (in place).
expand_defs(df, hed_schema, def_dict[, columns])	Expands any def tags found in the dataframe.
filter_series_by_onset(series, onsets)	Return the series, with rows that have the same onset combined.
process_def_expands(hed_strings, hed_schema)	Gather def-expand tags in the strings/compare with known definitions to find any differences.
replace_ref(text, oldvalue[, newvalue])	Replace column ref in x with y.
shrink_defs(df, hed_schema[, columns])	Shrink (in place) any def-expand tags found in the specified columns in the dataframe.
sort_dataframe_by_onsets(df)	Gather def-expand tags in the strings/compare with known definitions to find any differences.
split_delay_tags(series, hed_schema, onsets)	Sorts the series based on Delay tags, so that the onsets are in order after delay is applied.

convert_to_form(*df*, *hed_schema*, *tag_form*, *columns=None*)

Convert all tags in underlying dataframe to the specified form (in place).

Parameters

- **df** (*pd.Dataframe or pd.Series*) – The dataframe or series to modify.
- **hed_schema** (*HedSchema*) – The schema to use to convert tags.
- **tag_form** (*str*) – HedTag property to convert tags to.
- **columns** (*list*) – The columns to modify on the dataframe.

expand_defs(*df*, *hed_schema*, *def_dict*, *columns=None*)

Expands any def tags found in the dataframe.

Converts in place

Parameters

- **df** (*pd.Dataframe or pd.Series*) – The dataframe or series to modify.
- **hed_schema** (*HedSchema or None*) – The schema to use to identify defs.
- **def_dict** (*DefinitionDict*) – The definitions to expand.
- **columns** (*list or None*) – The columns to modify on the dataframe.

filter_series_by_onset(*series*, *onsets*)

Return the series, with rows that have the same onset combined.

Parameters

- **series** (*pd.Series or pd.Dataframe*) – the series to filter. If dataframe, it filters the "HED" column
- **onsets** (*pd.Series*) – the onset column to filter by

Returns

the series with rows filtered together.

Return type

Series or Dataframe

process_def_expands(*hed_strings*, *hed_schema*, *known_defs=None*, *ambiguous_defs=None*)

Gather def-expand tags in the strings/compare with known definitions to find any differences.

Parameters

- **hed_strings** (*list or pd.Series*) – A list of HED strings to process.
- **hed_schema** (*HedSchema*) – The schema to use.
- **known_defs** (*DefinitionDict or list or str or None*) – A DefinitionDict or anything its constructor takes. These are the known definitions going in, that must match perfectly.
- **ambiguous_defs** (*dict*) – A dictionary containing ambiguous definitions. format TBD. Currently def name key: list of lists of HED tags values

Returns

A tuple containing the DefinitionDict, ambiguous definitions, and errors.

Return type

tuple

replace_ref(*text*, *oldvalue*, *newvalue='n/a'*)

Replace column ref in x with y. If it's n/a, delete extra commas/parentheses.

Parameters

- **text** (*str*) – The input string containing the ref enclosed in curly braces.
- **oldvalue** (*str*) – The full tag or ref to replace
- **newvalue** (*str*) – The replacement value for the ref.

Returns

The modified string with the ref replaced or removed.

Return type

str

shrink_defs(*df*, *hed_schema*, *columns=None*)

Shrink (in place) any def-expand tags found in the specified columns in the dataframe.

Parameters

- **df** (*pd.Dataframe or pd.Series*) – The dataframe or series to modify.
- **hed_schema** (*HedSchema or None*) – The schema to use to identify defs.
- **columns** (*list or None*) – The columns to modify on the dataframe.

sort_dataframe_by_onsets(*df*)

Gather def-expand tags in the strings/compare with known definitions to find any differences.

Parameters

df (*pd.Dataframe*) – Dataframe to sort.

Returns

The sorted dataframe, or the original dataframe if it didn't have an onset column.

split_delay_tags(*series*, *hed_schema*, *onsets*)

Sorts the series based on Delay tags, so that the onsets are in order after delay is applied.

Parameters

- **series** (*pd.Series or None*) – the series of tags to split/sort

- **hed_schema** (*HedSchema*) – The schema to use to identify tags
- **onsets** (*pd.Series or None*) –

Returns

If we had onsets, a dataframe with 3 columns
"HED": The HED strings(still str) "onset": the updated onsets "original_index": the original source line. Multiple lines can have the same original source line.

Return type
sorted_df(pd.Dataframe or None)

Note: This dataframe may be longer than the original series, but it will never be shorter.

3.2.10 hed_group

A single parenthesized HED string.

Classes

HedGroup([hed_string, startpos, endpos, ...])	A single parenthesized HED string.

3.2.10.1 HedGroup

class HedGroup(*hed_string=''*, *startpos=None*, *endpos=None*, *contents=None*)

A single parenthesized HED string.

Methods

HedGroup.__init__([hed_string, startpos, ...])	Return an empty HedGroup object.
HedGroup.append(tag_or_group)	Add a tag or group to this group.
HedGroup.casefold()	Convenience function, equivalent to str(self).casefold().
HedGroup.check_if_in_original(tag_or_group)	Check if the tag or group in original string.
HedGroup.copy()	Return a deep copy of this group.
HedGroup.find_def_tags([recursive, ...])	Find def and def-expand tags.
HedGroup.find_exact_tags(exact_tags[, ...])	Find the given tags.
HedGroup.find_placeholder_tag()	Return a placeholder tag, if present in this group.
HedGroup.find_tags(search_tags[, recursive, ...])	Find the base tags and their containing groups.
HedGroup.find_tags_with_term(term[, ...])	Find any tags that contain the given term.
HedGroup.find_wildcard_tags(search_tags[, ...])	Find the tags and their containing groups.
HedGroup.get_all_groups([also_return_depth])	Return HedGroups, including descendants and self.
HedGroup.get_all_tags()	Return HedTags, including descendants.
HedGroup.get_as_form(tag_attribute)	Get the string corresponding to the specified form.
HedGroup.get_as_indented([tag_attribute])	Return the string as a multiline indented format.
HedGroup.get_as_long()	Return this HedGroup as a long tag string.
HedGroup.get_as_short()	Return this HedGroup as a short tag string.
HedGroup.get_first_group()	Return the first group in this HED string or group.
HedGroup.get_original_hed_string()	Get the original HED string.
HedGroup.groups()	Return the direct child groups of this group.
HedGroup.lower()	Convenience function, equivalent to str(self).lower().
HedGroup.remove(items_to_remove)	Remove any tags/groups in items_to_remove.
HedGroup.replace(item_to_replace, new_contents)	Replace an existing tag or group.
HedGroup.sort()	Sort the tags and groups in this HedString in a consistent order.
HedGroup.sorted()	Return a sorted copy of this HED group
HedGroup.tags()	Return the direct child tags of this group.

Attributes

HedGroup.is_group	True if this is a parenthesized group.
HedGroup.span	Return the source span.

HedGroup.**__init__**(*hed_string=''*, *startpos=None*, *endpos=None*, *contents=None*)

Return an empty HedGroup object.

Parameters

- **hed_string** (*str or None*) – Source HED string for this group.
- **startpos** (*int or None*) – Starting index of group(including parentheses) in hed_string.
- **endpos** (*int or None*) – Position after the end (including parentheses) in hed_string.
- **contents** (*list or None*) – A list of HedTags and/or HedGroups that will be set as the contents of this group. Mostly used during definition expansion.

HedGroup.**append**(*tag_or_group*)

Add a tag or group to this group.

Parameters

tag_or_group (*HedTag or HedGroup*) – The new object to add to this group.

HedGroup.**casefold**()

Convenience function, equivalent to str(self).casefold().

HedGroup.**check_if_in_original**(*tag_or_group*)

Check if the tag or group in original string.

Parameters
tag_or_group (*HedTag or HedGroup*) – The HedTag or HedGroup to be looked for in this group.

Returns
True if in this group.

Return type
bool

HedGroup.**copy**()

Return a deep copy of this group.

Returns
The copied group.

Return type
HedGroup

HedGroup.**find_def_tags**(*recursive=False*, *include_groups=3*)

Find def and def-expand tags.

Parameters

- **recursive** (*bool*) – If true, also check subgroups.
- **include_groups** (*int, 0, 1, 2, 3*) – Options for return values. If 0: Return only def and def expand tags/. If 1: Return only def tags and def-expand groups. If 2: Return only groups containing defs, or def-expand groups. If 3 or any other value: Return all 3 as a tuple.

Returns
A list of tuples. The contents depend on the values of the include_group.

Return type
list

HedGroup.**find_exact_tags**(*exact_tags*, *recursive=False*, *include_groups=1*)

Find the given tags. This will only find complete matches, any extension or value must also match.

Parameters

- **exact_tags** (*list of HedTag*) – A container of tags to locate.
- **recursive** (*bool*) – If true, also check subgroups.
- **include_groups** (*bool*) – 0, 1 or 2. If 0: Return only tags If 1: Return only groups If 2 or any other value: Return both

Returns
A list of tuples. The contents depend on the values of the include_group.

Return type
list

HedGroup.**find_placeholder_tag**()

Return a placeholder tag, if present in this group.

Returns
The placeholder tag if found.

Return type
HedTag or None

Notes

- Assumes a valid HedString with no erroneous "#" characters.

HedGroup.**find_tags**(*search_tags*, *recursive=False*, *include_groups=2*)

Find the base tags and their containing groups. This searches by short_base_tag, ignoring any ancestors or extensions/values.

Parameters

- **search_tags** (*container*) – A container of short_base_tags to locate.
- **recursive** (*bool*) – If true, also check subgroups.
- **include_groups** (*0, 1 or 2*) – Specify return values. If 0: return a list of the HedTags. If 1: return a list of the HedGroups containing the HedTags. If 2: return a list of tuples (HedTag, HedGroup) for the found tags.

Returns
The contents of the list depends on the value of include_groups.

Return type
list

HedGroup.**find_tags_with_term**(*term*, *recursive=False*, *include_groups=2*)

Find any tags that contain the given term.

Note: This can only find identified tags.

Parameters

- **term** (*str*) – A single term to search for.
- **recursive** (*bool*) – If true, recursively check subgroups.
- **include_groups** (*0, 1 or 2*) – Controls return values If 0: Return only tags. If 1: Return only groups. If 2 or any other value: Return both.

Return type
list

HedGroup.**find_wildcard_tags**(*search_tags*, *recursive=False*, *include_groups=2*)

Find the tags and their containing groups.

This searches tag.short_tag.casefold(), with an implicit wildcard on the end.

e.g. "Eve" will find Event, but not Sensory-event.

Parameters

- **search_tags** (*container*) – A container of the starts of short tags to search.
- **recursive** (*bool*) – If True, also check subgroups.

- **include_groups** (*0, 1 or 2*) – Specify return values. If 0: return a list of the HedTags. If 1: return a list of the HedGroups containing the HedTags. If 2: return a list of tuples (HedTag, HedGroup) for the found tags.

Returns
The contents of the list depends on the value of include_groups.

Return type
list

HedGroup.**get_all_groups**(*also_return_depth=False*)

Return HedGroups, including descendants and self.

Parameters
also_return_depth (*bool*) – If True, yield tuples (group, depth) rather than just groups.

Returns
The list of all HedGroups in this group, including descendants and self.

Return type
list

HedGroup.**get_all_tags**()

Return HedTags, including descendants.

Returns
A list of all the tags in this group including descendants.

Return type
list

HedGroup.**get_as_form**(*tag_attribute*)

Get the string corresponding to the specified form.

Parameters
tag_attribute (*str*) – The hed_tag property to use to construct the string (usually short_tag or long_tag).

Returns
The constructed string after transformation.

Return type
str

HedGroup.**get_as_indented**(*tag_attribute='short_tag'*)

Return the string as a multiline indented format.

Parameters
tag_attribute (*str*) – The hed_tag property to use to construct the string (usually short_tag or long_tag).

Returns
The indented string.

Return type
formatted_hed (str)

HedGroup.**get_as_long**()

Return this HedGroup as a long tag string.

Returns
The group as a string with all tags as long tags.

Return type
str

HedGroup.**get_as_short**()

Return this HedGroup as a short tag string.

Returns
The group as a string with all tags as short tags.

Return type
str

HedGroup.**get_first_group**()

Return the first group in this HED string or group.

Useful for things like Def-expand where they only have a single group.

Raises a ValueError if there are no groups.

Returns
The first group.

Return type
HedGroup

HedGroup.**get_original_hed_string**()

Get the original HED string.

Returns
The original string with no modification.

Return type
str

HedGroup.**groups**()

Return the direct child groups of this group.

Returns
All groups directly in this group, filtering out HedTag children.

Return type
list

HedGroup.**lower**()

Convenience function, equivalent to str(self).lower().

HedGroup.**remove**(*items_to_remove: Iterable[Union[HedTag, HedGroup]]*)

Remove any tags/groups in items_to_remove.

Parameters
items_to_remove (*list*) – List of HedGroups and/or HedTags to remove by identity.

Notes

- Any groups that become empty will also be pruned.
- If you pass a child and parent group, the child will also be removed from the parent.

static HedGroup.**replace**(*item_to_replace*, *new_contents*)

Replace an existing tag or group.

Note: This is a static method that relies on the parent attribute of item_to_replace.

Parameters

- **item_to_replace** (*HedTag or HedGroup*) – The item to replace must exist or this will raise an error.
- **new_contents** (*HedTag or HedGroup*) – Replacement contents.

Raises

- **KeyError** –
  - item_to_replace does not exist.
- **AttributeError** –
  - item_to_replace has no parent set.

HedGroup.**sort**()

Sort the tags and groups in this HedString in a consistent order.

HedGroup.**sorted**()

Return a sorted copy of this HED group

Returns
The sorted copy.

Return type
sorted_copy (HedGroup)

HedGroup.**tags**()

Return the direct child tags of this group.

Returns
All tags directly in this group, filtering out HedGroup children.

Return type
list

HedGroup.**is_group**

True if this is a parenthesized group.

HedGroup.**span**

Return the source span.

Returns
start index of the group (including parentheses) from the source string. int: end index of the group (including parentheses) from the source string.

Return type
int

3.2.11 hed_string

A HED string with its schema and definitions.

Classes

HedString(hed_string, hed_schema[, ...])	A HED string with its schema and definitions.

3.2.11.1 HedString

class HedString(*hed_string*, *hed_schema*, *def_dict=None*, *_contents=None*)

A HED string with its schema and definitions.

Methods

HedString.__init__(hed_string, hed_schema[, ...])	Constructor for the HedString class.
HedString.append(tag_or_group)	Add a tag or group to this group.
HedString.casefold()	Convenience function, equivalent to str(self).casefold().
HedString.check_if_in_original(tag_or_group)	Check if the tag or group in original string.
HedString.copy()	Return a deep copy of this string.
HedString.expand_defs()	Replace def tags with def-expand tags.
HedString.find_def_tags([recursive, ...])	Find def and def-expand tags.
HedString.find_exact_tags(exact_tags[, ...])	Find the given tags.
HedString.find_placeholder_tag()	Return a placeholder tag, if present in this group.
HedString.find_tags(search_tags[, ...])	Find the base tags and their containing groups.
HedString.find_tags_with_term(term[, ...])	Find any tags that contain the given term.
HedString.find_top_level_tags(anchor_tags[, ...])	Find top level groups with an anchor tag.
HedString.find_wildcard_tags(search_tags[, ...])	Find the tags and their containing groups.
HedString.from_hed_strings(hed_strings)	Factory for creating HedStrings via combination.
HedString.get_all_groups([also_return_depth])	Return HedGroups, including descendants and self.
HedString.get_all_tags()	Return HedTags, including descendants.
HedString.get_as_form(tag_attribute)	Get the string corresponding to the specified form.
HedString.get_as_indented([tag_attribute])	Return the string as a multiline indented format.
HedString.get_as_long()	Return this HedGroup as a long tag string.
HedString.get_as_original()	Return the original form of this string.
HedString.get_as_short()	Return this HedGroup as a short tag string.
HedString.get_first_group()	Return the first group in this HED string or group.
HedString.get_original_hed_string()	Get the original HED string.
HedString.groups()	Return the direct child groups of this group.
HedString.lower()	Convenience function, equivalent to str(self).lower().
HedString.remove(items_to_remove)	Remove any tags/groups in items_to_remove.
HedString.remove_definitions()	Remove definition tags and groups from this string.
HedString.remove_refs()	Remove any refs(tags contained entirely inside curly braces) from the string.
HedString.replace(item_to_replace, new_contents)	Replace an existing tag or group.
HedString.shrink_defs()	Replace def-expand tags with def tags.

continues on next page

Table 4 – continued from previous page

HedString.sort()	Sort the tags and groups in this HedString in a consistent order.
HedString.sorted()	Return a sorted copy of this HED group
HedString.split_hed_string(hed_string)	Split a HED string into delimiters and tags.
HedString.split_into_groups(hed_string, ...)	Split the HED string into a parse tree.
HedString.tags()	Return the direct child tags of this group.
HedString.validate([allow_placeholders, ...])	Validate the string using the schema.

Attributes

HedString.CLOSING_GROUP_CHARACTER	
HedString.OPENING_GROUP_CHARACTER	
HedString.is_group	Always False since the underlying string is not a group with parentheses.
HedString.span	Return the source span.

HedString.**__init__**(*hed_string*, *hed_schema*, *def_dict=None*, *_contents=None*)

Constructor for the HedString class.

Parameters

- **hed_string** (*str*) – A HED string consisting of tags and tag groups.
- **hed_schema** (*HedSchema*) – The schema to use to identify tags.
- **def_dict** (*DefinitionDict or None*) – The def dict to use to identify def/def expand tags.
- **_contents** (*[HedGroup and/or HedTag] or None*) – Create a HedString from this exact list of children. Does not make a copy.

Notes

- The HedString object parses its component tags and groups into a tree-like structure.

HedString.**append**(*tag_or_group*)

Add a tag or group to this group.

Parameters

tag_or_group (*HedTag or HedGroup*) – The new object to add to this group.

HedString.**casefold**()

Convenience function, equivalent to str(self).casefold().

HedString.**check_if_in_original**(*tag_or_group*)

Check if the tag or group in original string.

Parameters

tag_or_group (*HedTag or HedGroup*) – The HedTag or HedGroup to be looked for in this group.

Returns
True if in this group.

Return type
bool

HedString.**copy**()

Return a deep copy of this string.

Returns
The copied group.

Return type
HedString

HedString.**expand_defs**()

Replace def tags with def-expand tags.

This does very minimal validation.

Returns
self

HedString.**find_def_tags**(*recursive=False*, *include_groups=3*)

Find def and def-expand tags.

Parameters

- **recursive** (*bool*) – If true, also check subgroups.
- **include_groups** (*int, 0, 1, 2, 3*) – Options for return values. If 0: Return only def and def expand tags/. If 1: Return only def tags and def-expand groups. If 2: Return only groups containing defs, or def-expand groups. If 3 or any other value: Return all 3 as a tuple.

Returns
A list of tuples. The contents depend on the values of the include_group.

Return type
list

HedString.**find_exact_tags**(*exact_tags*, *recursive=False*, *include_groups=1*)

Find the given tags. This will only find complete matches, any extension or value must also match.

Parameters

- **exact_tags** (*list of HedTag*) – A container of tags to locate.
- **recursive** (*bool*) – If true, also check subgroups.
- **include_groups** (*bool*) – 0, 1 or 2. If 0: Return only tags If 1: Return only groups If 2 or any other value: Return both

Returns
A list of tuples. The contents depend on the values of the include_group.

Return type
list

HedString.**find_placeholder_tag**()

Return a placeholder tag, if present in this group.

Returns
The placeholder tag if found.

Return type
HedTag or None

Notes

- Assumes a valid HedString with no erroneous "#" characters.

HedString.**find_tags**(*search_tags*, *recursive=False*, *include_groups=2*)

Find the base tags and their containing groups. This searches by short_base_tag, ignoring any ancestors or extensions/values.

Parameters

- **search_tags** (*container*) – A container of short_base_tags to locate.
- **recursive** (*bool*) – If true, also check subgroups.
- **include_groups** (*0, 1 or 2*) – Specify return values. If 0: return a list of the HedTags. If 1: return a list of the HedGroups containing the HedTags. If 2: return a list of tuples (HedTag, HedGroup) for the found tags.

Returns
The contents of the list depends on the value of include_groups.

Return type
list

HedString.**find_tags_with_term**(*term*, *recursive=False*, *include_groups=2*)

Find any tags that contain the given term.

Note: This can only find identified tags.

Parameters

- **term** (*str*) – A single term to search for.
- **recursive** (*bool*) – If true, recursively check subgroups.
- **include_groups** (*0, 1 or 2*) – Controls return values If 0: Return only tags. If 1: Return only groups. If 2 or any other value: Return both.

Return type
list

HedString.**find_top_level_tags**(*anchor_tags*, *include_groups=2*)

Find top level groups with an anchor tag.

A max of 1 tag located per top level group.

Parameters

- **anchor_tags** (*container*) – A list/set/etc. of short_base_tags to find groups by.
- **include_groups** (*0, 1 or 2*) – Parameter indicating what return values to include. If 0: return only tags. If 1: return only groups. If 2 or any other value: return both.

Returns
The returned result depends on include_groups.

Return type
list

HedString.**find_wildcard_tags**(*search_tags*, *recursive=False*, *include_groups=2*)

Find the tags and their containing groups.

This searches tag.short_tag.casefold(), with an implicit wildcard on the end.

e.g. "Eve" will find Event, but not Sensory-event.

Parameters

- **search_tags** (*container*) – A container of the starts of short tags to search.
- **recursive** (*bool*) – If True, also check subgroups.
- **include_groups** (*0, 1 or 2*) – Specify return values. If 0: return a list of the HedTags. If 1: return a list of the HedGroups containing the HedTags. If 2: return a list of tuples (HedTag, HedGroup) for the found tags.

Returns
The contents of the list depends on the value of include_groups.

Return type
list

classmethod HedString.**from_hed_strings**(*hed_strings*)

Factory for creating HedStrings via combination.

Parameters
hed_strings (*list or None*) – A list of HedString objects to combine. This takes ownership of their children.

Returns
The newly combined HedString.

Return type
new_string(HedString)

HedString.**get_all_groups**(*also_return_depth=False*)

Return HedGroups, including descendants and self.

Parameters
also_return_depth (*bool*) – If True, yield tuples (group, depth) rather than just groups.

Returns
The list of all HedGroups in this group, including descendants and self.

Return type
list

HedString.**get_all_tags**()

Return HedTags, including descendants.

Returns
A list of all the tags in this group including descendants.

Return type
list

HedString.**get_as_form**(*tag_attribute*)

Get the string corresponding to the specified form.

Parameters
tag_attribute (*str*) – The hed_tag property to use to construct the string (usually short_tag or long_tag).

Returns
The constructed string after transformation.

Return type
str

HedString.**get_as_indented**(*tag_attribute='short_tag'*)

Return the string as a multiline indented format.

Parameters
tag_attribute (*str*) – The hed_tag property to use to construct the string (usually short_tag or long_tag).

Returns
The indented string.

Return type
formatted_hed (str)

HedString.**get_as_long**()

Return this HedGroup as a long tag string.

Returns
The group as a string with all tags as long tags.

Return type
str

HedString.**get_as_original**()

Return the original form of this string.

Returns
The string with all the tags in their original form.

Return type
str

Notes

Potentially with some extraneous spaces removed on returned string.

HedString.**get_as_short**()

Return this HedGroup as a short tag string.

Returns
The group as a string with all tags as short tags.

Return type
str

HedString.**get_first_group**()

Return the first group in this HED string or group.

Useful for things like Def-expand where they only have a single group.

Raises a ValueError if there are no groups.

Returns
The first group.

Return type
HedGroup

HedString.**get_original_hed_string**()

Get the original HED string.

Returns
The original string with no modification.

Return type
str

HedString.**groups**()

Return the direct child groups of this group.

Returns
All groups directly in this group, filtering out HedTag children.

Return type
list

HedString.**lower**()

Convenience function, equivalent to str(self).lower().

HedString.**remove**(*items_to_remove: Iterable[Union[HedTag, HedGroup]]*)

Remove any tags/groups in items_to_remove.

Parameters
items_to_remove (*list*) – List of HedGroups and/or HedTags to remove by identity.

Notes

- Any groups that become empty will also be pruned.
- If you pass a child and parent group, the child will also be removed from the parent.

HedString.**remove_definitions**()

Remove definition tags and groups from this string.

This does not validate definitions and will blindly removing invalid ones as well.

HedString.**remove_refs**()

Remove any refs(tags contained entirely inside curly braces) from the string.

This does NOT validate the contents of the curly braces. This is only relevant when directly editing sidecar strings. Tools will naturally ignore these.

static HedString.**replace**(*item_to_replace*, *new_contents*)

Replace an existing tag or group.

Note: This is a static method that relies on the parent attribute of item_to_replace.

Parameters

- **item_to_replace** (*HedTag or HedGroup*) – The item to replace must exist or this will raise an error.

- **new_contents** (*HedTag or HedGroup*) – Replacement contents.

Raises

- **KeyError** –
  - item_to_replace does not exist.
- **AttributeError** –
  - item_to_replace has no parent set.

HedString.**shrink_defs**()

Replace def-expand tags with def tags.

This does not validate them and will blindly shrink invalid ones as well.

Returns
self

HedString.**sort**()

Sort the tags and groups in this HedString in a consistent order.

HedString.**sorted**()

Return a sorted copy of this HED group

Returns
The sorted copy.

Return type
sorted_copy (HedGroup)

static HedString.**split_hed_string**(*hed_string*)

Split a HED string into delimiters and tags.

Parameters
hed_string (*str*) – The HED string to split.

Returns
A list of tuples where each tuple is (is_hed_tag, (start_pos, end_pos)).

Return type
list

Notes

- **The tuple format is as follows**
  - is_hed_tag (bool): A (possible) HED tag if True, delimiter if not.
  - start_pos (int): Index of start of string in hed_string.
  - end_pos (int): Index of end of string in hed_string.
- This function does not validate tags or delimiters in any form.

static HedString.**split_into_groups**(*hed_string*, *hed_schema*, *def_dict=None*)

Split the HED string into a parse tree.

Parameters

- **hed_string** (*str*) – A HED string consisting of tags and tag groups to be processed.

- **hed_schema** (*HedSchema*) – HED schema to use to identify tags.
- **def_dict** (*DefinitionDict*) – The definitions to identify.

Returns
A list of HedTag and/or HedGroup.

Return type
list

Raises
ValueError –

- The string is significantly malformed, such as mismatched parentheses.

Notes

- The parse tree consists of tag groups, tags, and delimiters.

HedString.**tags**()

Return the direct child tags of this group.

Returns
All tags directly in this group, filtering out HedGroup children.

Return type
list

HedString.**validate**(*allow_placeholders=True*, *error_handler=None*)

Validate the string using the schema.

Parameters

- **allow_placeholders** (*bool*) – Allow placeholders in the string.
- **error_handler** (*ErrorHandler or None*) – The error handler to use, creates a default one if none passed.

Returns
A list of issues for HED string.

Return type
issues (list of dict)

HedString.**CLOSING_GROUP_CHARACTER = ')'**

HedString.**OPENING_GROUP_CHARACTER = '('**

HedString.**is_group**

Always False since the underlying string is not a group with parentheses.

HedString.**span**

Return the source span.

Returns
start index of the group (including parentheses) from the source string. int: end index of the group (including parentheses) from the source string.

Return type
int

3.2.12 hed_tag

A single HED tag.

Classes

HedTag(hed_string, hed_schema[, span, def_dict])	A single HED tag.

3.2.12.1 HedTag

class HedTag(*hed_string*, *hed_schema*, *span=None*, *def_dict=None*)

A single HED tag.

Notes

- HedTag is a smart class in that it keeps track of its original value and positioning as well as pointers to the relevant HED schema information, if relevant.

Methods

HedTag.__init__(hed_string, hed_schema[, ...])	Creates a HedTag.
HedTag.base_tag_has_attribute(tag_attribute)	Check to see if the tag has a specific attribute.
HedTag.casefold()	Convenience function, equivalent to str(self).casefold().
HedTag.copy()	Return a deep copy of this tag.
HedTag.get_normalized_str()	
HedTag.get_stripped_unit_value(extension_text)	Return the extension divided into value and units, if the units are valid.
HedTag.get_tag_unit_class_units()	Get the unit class units associated with a particular tag.
HedTag.has_attribute(attribute)	Return True if this is an attribute this tag has.
HedTag.is_basic_tag()	Return True if a known tag with no extension or value.
HedTag.is_column_ref()	Return if this tag is a column reference from a sidecar.
HedTag.is_placeholder()	Returns if this tag has a placeholder in it.
HedTag.is_takes_value_tag()	Return True if this is a takes value tag.
HedTag.is_unit_class_tag()	Return True if this is a unit class tag.
HedTag.is_value_class_tag()	Return True if this is a value class tag.
HedTag.lower()	Convenience function, equivalent to str(self).lower().
HedTag.replace_placeholder(placeholder_value)	If tag has a placeholder character(#), replace with value.
HedTag.tag_exists_in_schema()	Return whether the schema entry for this tag exists.
HedTag.tag_modified()	Return True if tag has been modified from original.
HedTag.value_as_default_unit()	Return the value converted to default units if possible or None if invalid.

Attributes

HedTag.attributes	Return a dict of all the attributes this tag has.
HedTag.base_tag	Long form without value or extension.
HedTag.default_unit	Get the default unit class unit for this tag.
HedTag.expandable	Return what this expands to.
HedTag.expanded	Return if this is currently expanded or not.
HedTag.extension	Get the extension or value of tag.
HedTag.long_tag	Long form including value or extension.
HedTag.org_base_tag	Original form without value or extension.
HedTag.org_tag	Return the original unmodified tag.
HedTag.schema_namespace	Library namespace for this tag if one exists.
HedTag.short_base_tag	Short form without value or extension.
HedTag.short_tag	Short form including value or extension.
HedTag.tag	Returns the tag.
HedTag.unit_classes	Return a dict of all the unit classes this tag accepts.
HedTag.value_classes	Return a dict of all the value classes this tag accepts.

HedTag.**__init__**(*hed_string*, *hed_schema*, *span=None*, *def_dict=None*)

Creates a HedTag.

Parameters

- **hed_string** (*str*) – Source HED string for this tag.
- **hed_schema** (*HedSchema*) – A parameter for calculating canonical forms on creation.
- **span** (*int, int*) – The start and end indexes of the tag in the hed_string.
- **def_dict** (*DefinitionDict or None*) – The def dict to use to identify def/def expand tags.

HedTag.**base_tag_has_attribute**(*tag_attribute*)

Check to see if the tag has a specific attribute.

This is primarily used to check for things like TopLevelTag on Definitions and similar.

Parameters
tag_attribute (*str*) – A tag attribute.

Returns
True if the tag has the specified attribute. False, if otherwise.

Return type
bool

HedTag.**casefold**()

Convenience function, equivalent to str(self).casefold().

HedTag.**copy**()

Return a deep copy of this tag.

Returns
The copied group.

Return type
HedTag

HedTag.**get_normalized_str**()

HedTag.**get_stripped_unit_value**(*extension_text*)

Return the extension divided into value and units, if the units are valid.

Parameters
extension_text (*str*) – The text to split, in case it's a portion of a tag.

Returns
The extension portion with the units removed or None if invalid units. str or None: The units or None if no units of the right unit class are found.

Return type
str or None

Examples

'Duration/3 ms' will return ('3', 'ms')

HedTag.**get_tag_unit_class_units**()

Get the unit class units associated with a particular tag.

Returns
A list containing the unit class units associated with a particular tag or an empty list.

Return type
list

HedTag.**has_attribute**(*attribute*)

Return True if this is an attribute this tag has.

Parameters
attribute (*str*) – Name of the attribute.

Returns
True if this tag has the attribute.

Return type
bool

HedTag.**is_basic_tag**()

Return True if a known tag with no extension or value.

Returns
True if this is a known tag without extension or value.

Return type
bool

HedTag.**is_column_ref**()

Return if this tag is a column reference from a sidecar.

You should only see these if you are directly accessing sidecar strings, tools should remove them otherwise.

Returns
Returns True if this is a column ref.

Return type
bool

HedTag.**is_placeholder**()

Returns if this tag has a placeholder in it.

Returns
True if it has a placeholder

Return type
has_placeholder(bool)

HedTag.**is_takes_value_tag**()

Return True if this is a takes value tag.

Returns
True if this is a takes value tag.

Return type
bool

HedTag.**is_unit_class_tag**()

Return True if this is a unit class tag.

Returns
True if this is a unit class tag.

Return type
bool

HedTag.**is_value_class_tag**()

Return True if this is a value class tag.

Returns
True if this is a tag with a value class.

Return type
bool

HedTag.**lower**()

Convenience function, equivalent to str(self).lower().

HedTag.**replace_placeholder**(*placeholder_value*)

If tag has a placeholder character(#), replace with value.

Parameters
placeholder_value (*str*) – Value to replace placeholder with.

HedTag.**tag_exists_in_schema**()

Return whether the schema entry for this tag exists.

Returns
True if this tag exists.

Return type
bool

Notes

- This does NOT assure this is a valid tag.

HedTag.**tag_modified**()

Return True if tag has been modified from original.

Returns
Return True if the tag is modified.

Return type
bool

Notes

- Modifications can include adding a column name_prefix.

HedTag.**value_as_default_unit**()

Return the value converted to default units if possible or None if invalid.

Returns

The extension value in default units.
If no default units it assumes that the extension value is in default units.

Return type
float or None

Examples

'Duration/300 ms' will return .3

HedTag.**attributes**

Return a dict of all the attributes this tag has.

Returns empty dict if this is not a value tag.

Returns
A dict of attributes this tag has.

Return type
dict

Notes

- Returns empty dict if this is not a unit class tag.
- The dictionary has unit name as the key and HedSchemaEntry as value.

HedTag.**base_tag**

Long form without value or extension.

Returns
The long form of the tag, without value or extension.

Return type
base_tag (str)

HedTag.**default_unit**

Get the default unit class unit for this tag.

Only a tag with a single unit class can have default units.

Returns
the default unit entry for this tag, or None

Return type
unit(UnitEntry or None)

HedTag.**expandable**

Return what this expands to.

This is primarily used for Def/Def-expand tags at present.

Lazily set the first time it's called.

Returns
Returns the expanded form of this tag.

Return type
HedGroup or HedTag or None

HedTag.**expanded**

Return if this is currently expanded or not.

Will always be False unless expandable is set. This is primarily used for Def/Def-expand tags at present.

Returns
True if this is currently expanded.

Return type
bool

HedTag.**extension**

Get the extension or value of tag.

Generally this is just the portion after the last slash. Returns an empty string if no extension or value.

Returns
The tag name.

Return type
str

Notes

- This tag must have been computed first.

HedTag.**long_tag**

Long form including value or extension.

Returns
The long form of this tag.

Return type
str

HedTag.**org_base_tag**

Original form without value or extension.

Returns
The original form of the tag, without value or extension.

Return type
str

Notes

- Warning: This could be empty if the original tag had a name_prefix prepended. e.g. a column where "Label/" is prepended, thus the column value has zero base portion.

HedTag.**org_tag**

Return the original unmodified tag.

Returns
The original unmodified tag.

Return type
str

HedTag.**schema_namespace**

Library namespace for this tag if one exists.

Returns
The library namespace, including the colon.

Return type
namespace (str)

HedTag.**short_base_tag**

Short form without value or extension.

Returns
The short non-extension port of a tag.

Return type
str

Notes

- ParentNodes/Def/DefName would return just "Def".

HedTag.**short_tag**

Short form including value or extension.

Returns
The short form of the tag, including value or extension.

Return type
str

HedTag.**tag**

Returns the tag.

Returns the original tag if no user form set.

Returns
The custom set user form of the tag.

Return type
str

HedTag.**unit_classes**

Return a dict of all the unit classes this tag accepts.

Returns
A dict of unit classes this tag accepts.

Return type
dict

Notes

- Returns empty dict if this is not a unit class tag.
- The dictionary has unit name as the key and HedSchemaEntry as value.

HedTag.**value_classes**

Return a dict of all the value classes this tag accepts.

Returns
A dictionary of HedSchemaEntry value classes this tag accepts.

Return type
dict

Notes

- Returns empty dict if this is not a value class.
- The dictionary has unit name as the key and HedSchemaEntry as value.

3.2.13 model_constants

Defined constants for definitions, def labels, and expanded labels.

Classes

DefTagNames()	Source names for definitions, def labels, and expanded labels.

3.2.13.1 DefTagNames

class DefTagNames

Source names for definitions, def labels, and expanded labels.

Methods

DefTagNames.__init__()

Attributes

DefTagNames.ALL_TIME_KEYS
DefTagNames.DEFINITION_KEY
DefTagNames.DEF_EXPAND_KEY
DefTagNames.DEF_KEY
DefTagNames.DELAY_KEY
DefTagNames.DURATION_KEY
DefTagNames.DURATION_KEYS
DefTagNames.INSET_KEY
DefTagNames.OFFSET_KEY
DefTagNames.ONSET_KEY
DefTagNames.TEMPORAL_KEYS
DefTagNames.TIMELINE_KEYS

DefTagNames.**__init__**()

DefTagNames.**ALL_TIME_KEYS** = {'Delay', 'Duration', 'Inset', 'Offset', 'Onset'}

DefTagNames.**DEFINITION_KEY** = 'Definition'

DefTagNames.**DEF_EXPAND_KEY** = 'Def-expand'

DefTagNames.**DEF_KEY** = 'Def'

DefTagNames.**DELAY_KEY** = 'Delay'

DefTagNames.**DURATION_KEY** = 'Duration'

DefTagNames.**DURATION_KEYS** = {'Delay', 'Duration'}

DefTagNames.**INSET_KEY** = 'Inset'

DefTagNames.**OFFSET_KEY** = 'Offset'

DefTagNames.**ONSET_KEY** = 'Onset'

DefTagNames.**TEMPORAL_KEYS** = {'Inset', 'Offset', 'Onset'}

DefTagNames.**TIMELINE_KEYS** = {'Delay', 'Inset', 'Offset', 'Onset'}

3.2.14 query_expressions

Classes representing parsed query expressions.

Classes

Expression(token[, left, right])	Base class for parsed query expressions.
ExpressionAnd(token[, left, right])	
ExpressionDescendantGroup(token[, left, right])	
ExpressionExactMatch(token[, left, right])	
ExpressionNegation(token[, left, right])	
ExpressionOr(token[, left, right])	
ExpressionWildcardNew(token[, left, right])	

3.2.14.1 Expression

class Expression(*token*, *left=None*, *right=None*)

Base class for parsed query expressions.

Methods

Expression.__init__(token[, left, right])	
Expression.handle_expr(hed_group[, exact])	Handles parsing the given expression, recursively down the list as needed.

Attributes

Expression.**__init__**(*token*, *left=None*, *right=None*)

Expression.**handle_expr**(*hed_group*, *exact=False*)

Handles parsing the given expression, recursively down the list as needed.

BaseClass implementation is search terms.

Parameters

- **hed_group** (*HedGroup*) – The object to search
- **exact** (*bool*) – If True, we are only looking for groups containing this term directly, not descendants.

3.2.14.2 ExpressionAnd

class ExpressionAnd(*token*, *left=None*, *right=None*)

Methods

ExpressionAnd.__init__(token[, left, right])	
ExpressionAnd.handle_expr(hed_group[, exact])	Handles parsing the given expression, recursively down the list as needed.
ExpressionAnd.merge_and_groups(groups1, groups2)	Finds any shared results

Attributes

ExpressionAnd.**__init__**(*token*, *left=None*, *right=None*)

ExpressionAnd.**handle_expr**(*hed_group*, *exact=False*)

Handles parsing the given expression, recursively down the list as needed.

BaseClass implementation is search terms.

Parameters

- **hed_group** (*HedGroup*) – The object to search
- **exact** (*bool*) – If True, we are only looking for groups containing this term directly, not descendants.

static ExpressionAnd.**merge_and_groups**(*groups1*, *groups2*)

Finds any shared results

Parameters

- **groups1** (*list*) – a list of search results
- **groups2** (*list*) – a list of search results

Returns

groups in both lists narrowed down results to where none of the tags overlap

Return type

combined_groups(list)

3.2.14.3 ExpressionDescendantGroup

class ExpressionDescendantGroup(*token*, *left=None*, *right=None*)

Methods

ExpressionDescendantGroup.__init__(token[, ...])	
ExpressionDescendantGroup.handle_expr(hed_group)	Handles parsing the given expression, recursively down the list as needed.

Attributes

ExpressionDescendantGroup.**__init__**(*token*, *left=None*, *right=None*)

ExpressionDescendantGroup.**handle_expr**(*hed_group*, *exact=False*)

Handles parsing the given expression, recursively down the list as needed.

BaseClass implementation is search terms.

Parameters

- **hed_group** (*HedGroup*) – The object to search
- **exact** (*bool*) – If True, we are only looking for groups containing this term directly, not descendants.

3.2.14.4 ExpressionExactMatch

class ExpressionExactMatch(*token*, *left=None*, *right=None*)

Methods

ExpressionExactMatch.__init__(token[, left, ...])	
ExpressionExactMatch.handle_expr(hed_group)	Handles parsing the given expression, recursively down the list as needed.

Attributes

ExpressionExactMatch.**__init__**(*token*, *left=None*, *right=None*)

ExpressionExactMatch.**handle_expr**(*hed_group*, *exact=False*)

Handles parsing the given expression, recursively down the list as needed.

BaseClass implementation is search terms.

Parameters

- **hed_group** (*HedGroup*) – The object to search
- **exact** (*bool*) – If True, we are only looking for groups containing this term directly, not descendants.

3.2.14.5 ExpressionNegation

class ExpressionNegation(*token*, *left=None*, *right=None*)

Methods

ExpressionNegation.__init__(token[, left, right])	
ExpressionNegation.handle_expr(hed_group[, ...])	Handles parsing the given expression, recursively down the list as needed.

Attributes

ExpressionNegation.**__init__**(*token*, *left=None*, *right=None*)

ExpressionNegation.**handle_expr**(*hed_group*, *exact=False*)

Handles parsing the given expression, recursively down the list as needed.

BaseClass implementation is search terms.

Parameters

- **hed_group** (*HedGroup*) – The object to search
- **exact** (*bool*) – If True, we are only looking for groups containing this term directly, not descendants.

3.2.14.6 ExpressionOr

class ExpressionOr(*token*, *left=None*, *right=None*)

Methods

ExpressionOr.__init__(token[, left, right])	
ExpressionOr.handle_expr(hed_group[, exact])	Handles parsing the given expression, recursively down the list as needed.

Attributes

ExpressionOr.**__init__**(*token*, *left=None*, *right=None*)

ExpressionOr.**handle_expr**(*hed_group*, *exact=False*)

Handles parsing the given expression, recursively down the list as needed.

BaseClass implementation is search terms.

Parameters

- **hed_group** (*HedGroup*) – The object to search
- **exact** (*bool*) – If True, we are only looking for groups containing this term directly, not descendants.

3.2.14.7 ExpressionWildcardNew

class ExpressionWildcardNew(*token*, *left=None*, *right=None*)

Methods

ExpressionWildcardNew.__init__(token[, ...])	
ExpressionWildcardNew.handle_expr(hed_group)	Handles parsing the given expression, recursively down the list as needed.

Attributes

ExpressionWildcardNew.**__init__**(*token*, *left=None*, *right=None*)

ExpressionWildcardNew.**handle_expr**(*hed_group*, *exact=False*)

Handles parsing the given expression, recursively down the list as needed.

BaseClass implementation is search terms.

Parameters

- **hed_group** (*HedGroup*) – The object to search
- **exact** (*bool*) – If True, we are only looking for groups containing this term directly, not descendants.

3.2.15 query_handler

Holder for and manipulation of search results.

Classes

QueryHandler(expression_string)	Parse a search expression into a form than can be used to search a HED string.

3.2.15.1 QueryHandler

class QueryHandler(*expression_string*)

Parse a search expression into a form than can be used to search a HED string.

Methods

QueryHandler.__init__(expression_string)	Compiles a QueryHandler for a particular expression, so it can be used to search HED strings.
QueryHandler.search(hed_string_obj)	Returns if a match is found in the given string

Attributes

QueryHandler.**__init__**(*expression_string*)

Compiles a QueryHandler for a particular expression, so it can be used to search HED strings.

Basic Input Examples:

'Event' - Finds any strings with Event, or a descendent tag of Event such as Sensory-event.

'Event && Action' - Find any strings with Event and Action, including descendant tags.

'Event || Action' - Same as above, but it has either.

'"Event"' - Finds the Event tag, but not any descendent tags.

*Def/DefName/** - Find Def/DefName instances with placeholders, regardless of the value of the placeholder.

'Eve*' - Find any short tags that begin with Eve*, such as Event, but not Sensory-event.

'[Event && Action]' - Find a group that contains both Event and Action(at any level).

'{Event && Action}' - Find a group with Event And Action at the same level.

'{Event && Action:}' - Find a group with Event And Action at the same level, and nothing else.

'{Event && Action:Agent}' - Find a group with Event And Action at the same level, and optionally an Agent tag.

Practical Complex Example:

{(Onset || Offset), (Def || {Def-expand}): ???} - A group with an onset tag,
a def tag or def-expand group, and an optional wildcard group

Parameters
expression_string (*str*) – The query string.

QueryHandler.**search**(*hed_string_obj*)

Returns if a match is found in the given string

Parameters
hed_string_obj (*HedString*) – String to search

Returns

Generally you should just treat this as a bool
True if a match was found.

Return type
list(SearchResult)

3.2.16 query_service

Functions to get and use HED queries.

Functions

get_query_handlers(queries[, query_names])	Return a list of query handlers, query names, and issues if any.
search_hed_objs(hed_objs, queries, query_names)	Return a DataFrame of factors based on results of queries.

get_query_handlers(*queries*, *query_names=None*)

Return a list of query handlers, query names, and issues if any.

Parameters

- **queries** (*list*) – A list of query strings.
- **query_names** (*list or None*) – A list of column names for results of queries. If missing — query_1, query_2, etc.

Returns
list - QueryHandlers for successfully parsed queries. list - str names to assign to results of the queries. list - issues if any of the queries could not be parsed or other errors occurred.

search_hed_objs(*hed_objs*, *queries*, *query_names*)

Return a DataFrame of factors based on results of queries.

Parameters

- **hed_objs** (*list*) – A list of HedString objects (empty entries or None entries are 0's
- **queries** (*list*) – A list of query strings or QueryHandler objects.
- **query_names** (*list*) – A list of column names for results of queries.

Returns
Contains the factor vectors with results of the queries.

Return type
DataFrame

Raises
ValueError –

- If query names are invalid or duplicated.

3.2.17 query_util

Classes representing HED search results and tokens.

Classes

SearchResult(group, tag)	Holder for and manipulation of search results.
Token(text)	Represents a single term/character

3.2.17.1 SearchResult

class SearchResult(*group*, *tag*)

Holder for and manipulation of search results.

Methods

SearchResult.__init__(group, tag)	
SearchResult.has_same_tags(other)	Checks if these two results have the same tags/groups by identity(not equality)
SearchResult.merge_and_result(other)	Returns a new result, with the combined tags/groups from this and other.

Attributes

SearchResult.**__init__**(*group*, *tag*)

SearchResult.**has_same_tags**(*other*)

Checks if these two results have the same tags/groups by identity(not equality)

SearchResult.**merge_and_result**(*other*)

Returns a new result, with the combined tags/groups from this and other.

3.2.17.2 Token

class Token(*text*)

Represents a single term/character

Methods

Token.__init__(text)

Attributes

Token.And

Token.DescendantGroup

Token.DescendantGroupEnd

Token.ExactMatch

Token.ExactMatchEnd

Token.ExactMatchOptional

Token.LogicalGroup

Token.LogicalGroupEnd

Token.LogicalNegation

Token.NotInLine

Token.Or

Token.Tag

Token.Wildcard

Token.**__init__**(*text*)

Token.**And = 0**

Token.**DescendantGroup = 4**

Token.**DescendantGroupEnd = 5**

Token.**ExactMatch = 11**

Token.**ExactMatchEnd = 12**

Token.**ExactMatchOptional = 14**

Token.**LogicalGroup = 7**

Token.**LogicalGroupEnd = 8**

Token.**LogicalNegation = 9**

Token.**NotInLine = 13**

Token.**Or = 6**

Token.**Tag = 1**

Token.**Wildcard = 10**

3.2.18 sidecar

Contents of a JSON file or merged JSON files.

Classes

Sidecar(files[, name])	Contents of a JSON file or JSON files.

3.2.18.1 Sidecar

class Sidecar(*files*, *name=None*)

Contents of a JSON file or JSON files.

Methods

Sidecar.__init__(files[, name])	Construct a Sidecar object representing a JSON file.
Sidecar.extract_definitions(hed_schema[, ...])	Gather and validate definitions in metadata.
Sidecar.get_as_json_string()	Return this sidecar's column metadata as a string.
Sidecar.get_column_refs()	Returns a list of column refs found in this sidecar.
Sidecar.get_def_dict(hed_schema[, ...])	Return the definition dict for this sidecar.
Sidecar.load_sidecar_file(file)	Load column metadata from a given json file.
Sidecar.load_sidecar_files(files)	Load json from a given file or list.
Sidecar.save_as_json(save_filename)	Save column metadata to a JSON file.
Sidecar.validate(hed_schema[, ...])	Create a SidecarValidator and validate this sidecar with the schema.

Attributes

Sidecar.all_hed_columns	Return all columns that are HED compatible.
Sidecar.column_data	Generate the ColumnMetadata for this sidecar.
Sidecar.def_dict	Definitions from this sidecar.

Sidecar.**__init__**(*files*, *name=None*)

Construct a Sidecar object representing a JSON file.

Parameters

- **files** (*str or FileLike or list*) – A string or file-like object representing a JSON file, or a list of such.
- **name** (*str or None*) – Optional name identifying this sidecar, generally a filename.

Sidecar.**extract_definitions**(*hed_schema*, *error_handler=None*)

Gather and validate definitions in metadata.

Parameters

- **hed_schema** (*HedSchema*) – The schema to used to identify tags.
- **error_handler** (*ErrorHandler or None*) – The error handler to use for context, uses a default one if None.

Returns
Contains all the definitions located in the sidecar.

Return type
DefinitionDict

Sidecar.**get_as_json_string**()

Return this sidecar's column metadata as a string.

Returns
The json string representing this sidecar.

Return type
str

Sidecar.**get_column_refs**()

Returns a list of column refs found in this sidecar.

This does not validate

Returns
A list of unique column refs found.

Return type
column_refs(list)

Sidecar.**get_def_dict**(*hed_schema*, *extra_def_dicts=None*)

Return the definition dict for this sidecar.

Parameters

- **hed_schema** (*HedSchema*) – Identifies tags to find definitions.
- **extra_def_dicts** (*list, DefinitionDict, or None*) – Extra dicts to add to the list.

Returns
A single definition dict representing all the data(and extra def dicts).

Return type
DefinitionDict

Sidecar.**load_sidecar_file**(*file*)

Load column metadata from a given json file.

Parameters
file (*str or FileLike*) – If a string, this is a filename. Otherwise, it will be parsed as a file-like.

Raises
HedFileError –

- If the file was not found or could not be parsed into JSON.

Sidecar.**load_sidecar_files**(*files*)

Load json from a given file or list.

Parameters
files (*str or FileLike or list*) – A string or file-like object representing a JSON file, or a list of such.

Raises
HedFileError –

- If the file was not found or could not be parsed into JSON.

Sidecar.**save_as_json**(*save_filename*)

Save column metadata to a JSON file.

Parameters
save_filename (*str*) – Path to save file.

Sidecar.**validate**(*hed_schema*, *extra_def_dicts=None*, *name=None*, *error_handler=None*)

Create a SidecarValidator and validate this sidecar with the schema.

Parameters

- **hed_schema** (*HedSchema*) – Input data to be validated.
- **extra_def_dicts** (*list or DefinitionDict*) – Extra def dicts in addition to sidecar.
- **name** (*str*) – The name to report this sidecar as.
- **error_handler** (*ErrorHandler*) – Error context to use. Creates a new one if None.

Returns
A list of issues associated with each level in the HED string.

Return type
issues (list of dict)

Sidecar.**all_hed_columns**

Return all columns that are HED compatible.

Returns
A list of all valid HED columns by name.

Return type
column_refs(list)

Sidecar.**column_data**

Generate the ColumnMetadata for this sidecar.

Returns
ColumnMetadata}): The column metadata defined by this sidecar.

Return type
dict({str

Sidecar.**def_dict**

Definitions from this sidecar.

Generally you should instead call get_def_dict to get the relevant definitions.

Returns
The definitions for this sidecar.

Return type
DefinitionDict

3.2.19 spreadsheet_input

A spreadsheet of HED tags.

Classes

SpreadsheetInput([file, file_type, ...])	A spreadsheet of HED tags.

3.2.19.1 SpreadsheetInput

class SpreadsheetInput(*file=None*, *file_type=None*, *worksheet_name=None*, *tag_columns=None*, *has_column_names=True*, *column_prefix_dictionary=None*, *name=None*)

A spreadsheet of HED tags.

Methods

SpreadsheetInput.__init__([file, file_type, ...])	Constructor for the SpreadsheetInput class.
SpreadsheetInput.assemble([mapper, ...])	Assembles the HED strings.
SpreadsheetInput.column_metadata()	Return the metadata for each column.
SpreadsheetInput.combine_dataframe(dataframe)	Combine all columns in the given dataframe into a single HED string series,
SpreadsheetInput.convert_to_form(hed_schema, ...)	Convert all tags in underlying dataframe to the specified form.
SpreadsheetInput.convert_to_long(hed_schema)	Convert all tags in underlying dataframe to long form.
SpreadsheetInput.convert_to_short(hed_schema)	Convert all tags in underlying dataframe to short form.
SpreadsheetInput.expand_defs(hed_schema, ...)	Shrinks any def-expand found in the underlying dataframe.
SpreadsheetInput.get_column_refs()	Return a list of column refs for this file.
SpreadsheetInput.get_def_dict(hed_schema[, ...])	Return the definition dict for this file.
SpreadsheetInput.get_worksheet([worksheet_name])	Get the requested worksheet.
SpreadsheetInput.reset_mapper(new_mapper)	Set mapper to a different view of the file.
SpreadsheetInput.set_cell(row_number, ...[, ...])	Replace the specified cell with transformed text.
SpreadsheetInput.shrink_defs(hed_schema)	Shrinks any def-expand found in the underlying dataframe.
SpreadsheetInput.to_csv([file])	Write to file or return as a string.
SpreadsheetInput.to_excel(file)	Output to an Excel file.
SpreadsheetInput.validate(hed_schema[, ...])	Creates a SpreadsheetValidator and returns all issues with this file.

Attributes

SpreadsheetInput.EXCEL_EXTENSION	
SpreadsheetInput.TEXT_EXTENSION	
SpreadsheetInput.columns	Returns a list of the column names.
SpreadsheetInput.dataframe	The underlying dataframe.
SpreadsheetInput.dataframe_a	Return the assembled dataframe Probably a placeholder name.
SpreadsheetInput.has_column_names	True if dataframe has column names.
SpreadsheetInput.loaded_workbook	The underlying loaded workbooks.
SpreadsheetInput.name	Name of the data.
SpreadsheetInput.needs_sorting	Return True if this both has an onset column, and it needs sorting.
SpreadsheetInput.onsets	Return the onset column if it exists.
SpreadsheetInput.series_a	Return the assembled dataframe as a series.
SpreadsheetInput.series_filtered	Return the assembled dataframe as a series, with rows that have the same onset combined.
SpreadsheetInput.worksheet_name	The worksheet name.

SpreadsheetInput.**__init__**(*file=None*, *file_type=None*, *worksheet_name=None*, *tag_columns=None*, *has_column_names=True*, *column_prefix_dictionary=None*, *name=None*)

Constructor for the SpreadsheetInput class.

Parameters

- **file** (*str or file like*) – An xlsx/tsv file to open or a File object.
- **file_type** (*str or None*) – ".xlsx" for Excel, ".tsv" or ".txt" for tsv. data.
- **worksheet_name** (*str or None*) – The name of the Excel workbook worksheet that contains the HED tags. Not applicable to tsv files. If omitted for Excel, the first worksheet is assumed.
- **tag_columns** (*list*) – A list of ints or strs containing the columns that contain the HED tags. If ints then column numbers with [1] indicating only the second column has tags.
- **has_column_names** (*bool*) – True if file has column names. Validation will skip over the first row. first line of the file if the spreadsheet as column names.
- **column_prefix_dictionary** (*dict or None*) – Dictionary with keys that are column numbers/names and values are HED tag prefixes to prepend to the tags in that column before processing.

Notes

- If file is a string, file_type is derived from file and this parameter is ignored.
- column_prefix_dictionary may be deprecated/renamed. These are no longer prefixes, but rather converted to value columns. e.g. {"key": "Description", 1: "Label/"} will turn into value columns as {"key": "Description/#", 1: "Label/#"} It will be a validation issue if column 1 is called "key" in the above example. This means it no longer accepts anything but the value portion only in the columns.

Raises

HedFileError –

- The file is blank.
- An invalid dataframe was passed with size 0.
- An invalid extension was provided.
- A duplicate or empty column name appears.
- Cannot open the indicated file.
- The specified worksheet name does not exist.

SpreadsheetInput.**assemble**(*mapper=None*, *skip_curly_braces=False*)

Assembles the HED strings.

Parameters

- **mapper** (*ColumnMapper or None*) – Generally pass none here unless you want special behavior.
- **skip_curly_braces** (*bool*) – If True, don't plug in curly brace values into columns.

Returns

The assembled dataframe.

Return type

Dataframe

SpreadsheetInput.**column_metadata**()

Return the metadata for each column.

Returns
Number/ColumnMeta pairs.

Return type
dict

static SpreadsheetInput.**combine_dataframe**(*dataframe*)

Combine all columns in the given dataframe into a single HED string series,
skipping empty columns and columns with empty strings.

Parameters
dataframe (*Dataframe*) – The dataframe to combine

Returns
The assembled series.

Return type
Series

SpreadsheetInput.**convert_to_form**(*hed_schema*, *tag_form*)

Convert all tags in underlying dataframe to the specified form.

Parameters

- **hed_schema** (*HedSchema*) – The schema to use to convert tags.
- **tag_form** (*str*) – HedTag property to convert tags to. Most cases should use convert_to_short or convert_to_long below.

SpreadsheetInput.**convert_to_long**(*hed_schema*)

Convert all tags in underlying dataframe to long form.

Parameters
hed_schema (*HedSchema or None*) – The schema to use to convert tags.

SpreadsheetInput.**convert_to_short**(*hed_schema*)

Convert all tags in underlying dataframe to short form.

Parameters
hed_schema (*HedSchema*) – The schema to use to convert tags.

SpreadsheetInput.**expand_defs**(*hed_schema*, *def_dict*)

Shrinks any def-expand found in the underlying dataframe.

Parameters

- **hed_schema** (*HedSchema or None*) – The schema to use to identify defs.
- **def_dict** (*DefinitionDict*) – The definitions to expand.

SpreadsheetInput.**get_column_refs**()

Return a list of column refs for this file.

Default implementation returns none.

Returns
A list of unique column refs found.

Return type
column_refs(list)

SpreadsheetInput.**get_def_dict**(*hed_schema*, *extra_def_dicts=None*)

Return the definition dict for this file.

Note: Baseclass implementation returns just extra_def_dicts.

Parameters

- **hed_schema** (*HedSchema*) – Identifies tags to find definitions(if needed).
- **extra_def_dicts** (*list, DefinitionDict, or None*) – Extra dicts to add to the list.

Returns
A single definition dict representing all the data(and extra def dicts).

Return type
DefinitionDict

SpreadsheetInput.**get_worksheet**(*worksheet_name=None*)

Get the requested worksheet.

Parameters
worksheet_name (*str or None*) – The name of the requested worksheet by name or the first one if None.

Returns
The workbook request.

Return type
openpyxl.workbook.Workbook

Notes

If None, returns the first worksheet.

Raises
KeyError –

- The specified worksheet name does not exist.

SpreadsheetInput.**reset_mapper**(*new_mapper*)

Set mapper to a different view of the file.

Parameters
new_mapper (*ColumnMapper*) – A column mapper to be associated with this base input.

SpreadsheetInput.**set_cell**(*row_number*, *column_number*, *new_string_obj*, *tag_form='short_tag'*)

Replace the specified cell with transformed text.

Parameters

- **row_number** (*int*) – The row number of the spreadsheet to set.
- **column_number** (*int*) – The column number of the spreadsheet to set.
- **new_string_obj** (*HedString*) – Object with text to put in the given cell.
- **tag_form** (*str*) – Version of the tags (short_tag, long_tag, base_tag, etc.)

Notes

Any attribute of a HedTag that returns a string is a valid value of tag_form.

Raises

- **ValueError** –
  - There is not a loaded dataframe.
- **KeyError** –
  - The indicated row/column does not exist.
- **AttributeError** –
  - The indicated tag_form is not an attribute of HedTag.

SpreadsheetInput.**shrink_defs**(*hed_schema*)

Shrinks any def-expand found in the underlying dataframe.

Parameters
hed_schema (*HedSchema or None*) – The schema to use to identify defs.

SpreadsheetInput.**to_csv**(*file=None*)

Write to file or return as a string.

Parameters
file (*str, file-like, or None*) – Location to save this file. If None, return as string.

Returns
None if file is given or the contents as a str if file is None.

Return type
None or str

Raises
OSError –
- Cannot open the indicated file.

SpreadsheetInput.**to_excel**(*file*)

Output to an Excel file.

Parameters
file (*str or file-like*) – Location to save this base input.

Raises

- **ValueError** –
  - If empty file object was passed.
- **OSError** –
  - Cannot open the indicated file.

SpreadsheetInput.**validate**(*hed_schema*, *extra_def_dicts=None*, *name=None*, *error_handler=None*)

Creates a SpreadsheetValidator and returns all issues with this file.

Parameters

- **hed_schema** (*HedSchema*) – The schema to use for validation.
- **extra_def_dicts** (*list of DefDict or DefDict*) – All definitions to use for validation.

- **name** (*str*) – The name to report errors from this file as.
- **error_handler** (*ErrorHandler*) – Error context to use. Creates a new one if None.

Returns
A list of issues for a HED string.

Return type
issues (list of dict)

SpreadsheetInput.**EXCEL_EXTENSION = ['.xlsx']**

SpreadsheetInput.**TEXT_EXTENSION = ['.tsv', '.txt']**

SpreadsheetInput.**columns**

Returns a list of the column names.

Empty if no column names.

Returns
The column names.

Return type
columns(list)

SpreadsheetInput.**dataframe**

The underlying dataframe.

SpreadsheetInput.**dataframe_a**

Return the assembled dataframe Probably a placeholder name.

Returns
the assembled dataframe

Return type
Dataframe

SpreadsheetInput.**has_column_names**

True if dataframe has column names.

SpreadsheetInput.**loaded_workbook**

The underlying loaded workbooks.

SpreadsheetInput.**name**

Name of the data.

SpreadsheetInput.**needs_sorting**

Return True if this both has an onset column, and it needs sorting.

SpreadsheetInput.**onsets**

Return the onset column if it exists.

SpreadsheetInput.**series_a**

Return the assembled dataframe as a series.

Returns
the assembled dataframe with columns merged.

Return type
Series

SpreadsheetInput.**series_filtered**

Return the assembled dataframe as a series, with rows that have the same onset combined.

Returns
the assembled dataframe with columns merged, and the rows filtered together.

Return type
Series or None

SpreadsheetInput.**worksheet_name**

The worksheet name.

3.2.20 string_util

Utilities for manipulating HedString objects.

Functions

cleanup_empties(string_in)	
gather_descriptions(hed_string)	Remove any description tags from the HedString and concatenates them.
split_base_tags(hed_string, base_tags[, ...])	Split a HedString object into two separate HedString objects based on the presence of base tags.
split_def_tags(hed_string, def_names[, ...])	Split a HedString object into two separate HedString objects based on the presence of def tags

cleanup_empties(*string_in: str*) → str

gather_descriptions(*hed_string*)

Remove any description tags from the HedString and concatenates them.

Parameters
hed_string (*HedString*) – To be modified.

Returns: tuple
description(str): The concatenated values of all description tags.

Side effect:
The input HedString has its description tags removed.

split_base_tags(*hed_string*, *base_tags*, *remove_group=False*)

Split a HedString object into two separate HedString objects based on the presence of base tags.

Parameters

- **hed_string** (*HedString*) – The input HedString object to be split.
- **base_tags** (*list of str*) – A list of strings representing the base tags. This is matching the base tag NOT all the terms above it.
- **remove_group** (*bool, optional*) – Flag indicating whether to remove the parent group. Defaults to False.

Returns

A tuple containing two HedString objects:

- The first HedString object contains the remaining tags from hed_string.
- The second HedString object contains the tags from hed_string that match the base_tags.

Return type
tuple

`split_def_tags`(*hed_string*, *def_names*, *remove_group=False*)

Split a HedString object into two separate HedString objects based on the presence of def tags

This does NOT handle def-expand tags currently.

Parameters

- **hed_string** (*HedString*) – The input HedString object to be split.
- **def_names** (*list of str*) – A list of def names to search for. Can optionally include a value.
- **remove_group** (*bool, optional*) – Flag indicating whether to remove the parent group. Defaults to False.

Returns

A tuple containing two HedString objects:

- The first HedString object contains the remaining tags from hed_string.
- The second HedString object contains the tags from hed_string that match the def_names.

Return type
tuple

3.2.21 tabular_input

A BIDS tabular file with sidecar.

Classes

`TabularInput`([file, sidecar, name])	A BIDS tabular file with sidecar.

3.2.21.1 TabularInput

`class TabularInput`(*file=None*, *sidecar=None*, *name=None*)

A BIDS tabular file with sidecar.

Methods

TabularInput.__init__([file, sidecar, name])	Constructor for the TabularInput class.
TabularInput.assemble([mapper, ...])	Assembles the HED strings.
TabularInput.column_metadata()	Return the metadata for each column.
TabularInput.combine_dataframe(dataframe)	Combine all columns in the given dataframe into a single HED string series,
TabularInput.convert_to_form(hed_schema, ...)	Convert all tags in underlying dataframe to the specified form.
TabularInput.convert_to_long(hed_schema)	Convert all tags in underlying dataframe to long form.
TabularInput.convert_to_short(hed_schema)	Convert all tags in underlying dataframe to short form.
TabularInput.expand_defs(hed_schema, def_dict)	Shrinks any def-expand found in the underlying dataframe.
TabularInput.get_column_refs()	Return a list of column refs for this file.
TabularInput.get_def_dict(hed_schema[, ...])	Return the definition dict for this sidecar.
TabularInput.get_sidecar()	Return the sidecar associated with this TabularInput.
TabularInput.get_worksheet([worksheet_name])	Get the requested worksheet.
TabularInput.reset_column_mapper([sidecar])	Change the sidecars and settings.
TabularInput.reset_mapper(new_mapper)	Set mapper to a different view of the file.
TabularInput.set_cell(row_number, ...[, ...])	Replace the specified cell with transformed text.
TabularInput.shrink_defs(hed_schema)	Shrinks any def-expand found in the underlying dataframe.
TabularInput.to_csv([file])	Write to file or return as a string.
TabularInput.to_excel(file)	Output to an Excel file.
TabularInput.validate(hed_schema[, ...])	Creates a SpreadsheetValidator and returns all issues with this file.

Attributes

TabularInput.EXCEL_EXTENSION	
TabularInput.HED_COLUMN_NAME	
TabularInput.TEXT_EXTENSION	
TabularInput.columns	Returns a list of the column names.
TabularInput.dataframe	The underlying dataframe.
TabularInput.dataframe_a	Return the assembled dataframe Probably a placeholder name.
TabularInput.has_column_names	True if dataframe has column names.
TabularInput.loaded_workbook	The underlying loaded workbooks.
TabularInput.name	Name of the data.
TabularInput.needs_sorting	Return True if this both has an onset column, and it needs sorting.
TabularInput.onsets	Return the onset column if it exists.
TabularInput.series_a	Return the assembled dataframe as a series.
TabularInput.series_filtered	Return the assembled dataframe as a series, with rows that have the same onset combined.
TabularInput.worksheet_name	The worksheet name.

TabularInput.__init__(*file=None*, *sidecar=None*, *name=None*)

Constructor for the TabularInput class.

Parameters

- **file** (*str or FileLike or pd.Dataframe*) – A tsv file to open.
- **sidecar** (*str or Sidecar or FileLike*) – A Sidecar or source file/filename.
- **name** (*str*) – The name to display for this file for error purposes.

Raises

- *HedFileError* –
  - The file is blank.
  - An invalid dataframe was passed with size 0.
  - An invalid extension was provided.
  - A duplicate or empty column name appears.
- **OSError** –
  - Cannot open the indicated file.
- **ValueError** –
  - This file has no column names.

TabularInput.assemble(*mapper=None*, *skip_curly_braces=False*)

Assembles the HED strings.

Parameters

- **mapper** (*ColumnMapper or None*) – Generally pass none here unless you want special behavior.
- **skip_curly_braces** (*bool*) – If True, don't plug in curly brace values into columns.

Returns
The assembled dataframe.

Return type
Dataframe

TabularInput.column_metadata()

Return the metadata for each column.

Returns
Number/ColumnMeta pairs.

Return type
dict

static TabularInput.combine_dataframe(*dataframe*)

Combine all columns in the given dataframe into a single HED string series,
skipping empty columns and columns with empty strings.

Parameters
dataframe (*Dataframe*) – The dataframe to combine

Returns
The assembled series.

Return type
Series

TabularInput.**convert_to_form**(*hed_schema*, *tag_form*)

Convert all tags in underlying dataframe to the specified form.

Parameters

- **hed_schema** (*HedSchema*) – The schema to use to convert tags.
- **tag_form** (*str*) – HedTag property to convert tags to. Most cases should use convert_to_short or convert_to_long below.

TabularInput.**convert_to_long**(*hed_schema*)

Convert all tags in underlying dataframe to long form.

Parameters
hed_schema (*HedSchema or None*) – The schema to use to convert tags.

TabularInput.**convert_to_short**(*hed_schema*)

Convert all tags in underlying dataframe to short form.

Parameters
hed_schema (*HedSchema*) – The schema to use to convert tags.

TabularInput.**expand_defs**(*hed_schema*, *def_dict*)

Shrinks any def-expand found in the underlying dataframe.

Parameters

- **hed_schema** (*HedSchema or None*) – The schema to use to identify defs.
- **def_dict** (*DefinitionDict*) – The definitions to expand.

TabularInput.**get_column_refs**()

Return a list of column refs for this file.

Default implementation returns none.

Returns
A list of unique column refs found.

Return type
column_refs(list)

TabularInput.**get_def_dict**(*hed_schema*, *extra_def_dicts=None*)

Return the definition dict for this sidecar.

Parameters

- **hed_schema** (*HedSchema*) – Used to identify tags to find definitions.
- **extra_def_dicts** (*list, DefinitionDict, or None*) – Extra dicts to add to the list.

Returns
A single definition dict representing all the data(and extra def dicts).

Return type
DefinitionDict

TabularInput.**get_sidecar**()

Return the sidecar associated with this TabularInput.

TabularInput.**get_worksheet**(*worksheet_name=None*)

Get the requested worksheet.

Parameters
worksheet_name (*str or None*) – The name of the requested worksheet by name or the first one if None.

Returns
The workbook request.

Return type
openpyxl.workbook.Workbook

Notes

If None, returns the first worksheet.

Raises
KeyError –

- The specified worksheet name does not exist.

TabularInput.**reset_column_mapper**(*sidecar=None*)

Change the sidecars and settings.

Parameters
sidecar (*str or [str] or Sidecar or [Sidecar]*) – A list of json filenames to pull sidecar info from.

TabularInput.**reset_mapper**(*new_mapper*)

Set mapper to a different view of the file.

Parameters
new_mapper (*ColumnMapper*) – A column mapper to be associated with this base input.

TabularInput.**set_cell**(*row_number*, *column_number*, *new_string_obj*, *tag_form='short_tag'*)

Replace the specified cell with transformed text.

Parameters

- **row_number** (*int*) – The row number of the spreadsheet to set.
- **column_number** (*int*) – The column number of the spreadsheet to set.
- **new_string_obj** (*HedString*) – Object with text to put in the given cell.
- **tag_form** (*str*) – Version of the tags (short_tag, long_tag, base_tag, etc.)

Notes

Any attribute of a HedTag that returns a string is a valid value of tag_form.

Raises

- **ValueError** –
  - There is not a loaded dataframe.
- **KeyError** –
  - The indicated row/column does not exist.

- **AttributeError** –
  - The indicated tag_form is not an attribute of HedTag.

TabularInput.**shrink_defs**(*hed_schema*)

Shrinks any def-expand found in the underlying dataframe.

Parameters
hed_schema (*HedSchema or None*) – The schema to use to identify defs.

TabularInput.**to_csv**(*file=None*)

Write to file or return as a string.

Parameters
file (*str, file-like, or None*) – Location to save this file. If None, return as string.

Returns
None if file is given or the contents as a str if file is None.

Return type
None or str

Raises
OSError –
- Cannot open the indicated file.

TabularInput.**to_excel**(*file*)

Output to an Excel file.

Parameters
file (*str or file-like*) – Location to save this base input.

Raises
- **ValueError** –
  - If empty file object was passed.
- **OSError** –
  - Cannot open the indicated file.

TabularInput.**validate**(*hed_schema*, *extra_def_dicts=None*, *name=None*, *error_handler=None*)

Creates a SpreadsheetValidator and returns all issues with this file.

Parameters
- **hed_schema** (*HedSchema*) – The schema to use for validation.
- **extra_def_dicts** (*list of DefDict or DefDict*) – All definitions to use for validation.
- **name** (*str*) – The name to report errors from this file as.
- **error_handler** (*ErrorHandler*) – Error context to use. Creates a new one if None.

Returns
A list of issues for a HED string.

Return type
issues (list of dict)

TabularInput.**EXCEL_EXTENSION = ['.xlsx']**

TabularInput.**HED_COLUMN_NAME = 'HED'**

TabularInput.**TEXT_EXTENSION = ['.tsv', '.txt']**

TabularInput.**columns**

Returns a list of the column names.

Empty if no column names.

Returns
The column names.

Return type
columns(list)

TabularInput.**dataframe**

The underlying dataframe.

TabularInput.**dataframe_a**

Return the assembled dataframe Probably a placeholder name.

Returns
the assembled dataframe

Return type
Dataframe

TabularInput.**has_column_names**

True if dataframe has column names.

TabularInput.**loaded_workbook**

The underlying loaded workbooks.

TabularInput.**name**

Name of the data.

TabularInput.**needs_sorting**

Return True if this both has an onset column, and it needs sorting.

TabularInput.**onsets**

Return the onset column if it exists.

TabularInput.**series_a**

Return the assembled dataframe as a series.

Returns
the assembled dataframe with columns merged.

Return type
Series

TabularInput.**series_filtered**

Return the assembled dataframe as a series, with rows that have the same onset combined.

Returns
the assembled dataframe with columns merged, and the rows filtered together.

Return type
Series or None

TabularInput.**worksheet_name**

The worksheet name.

3.2.22 timeseries_input

A BIDS time series tabular file.

Classes

TimeseriesInput([file, sidecar, ...])	A BIDS time series tabular file.

3.2.22.1 TimeseriesInput

class TimeseriesInput(*file=None*, *sidecar=None*, *extra_def_dicts=None*, *name=None*)

A BIDS time series tabular file.

Methods

TimeseriesInput.__init__([file, sidecar, ...])	Constructor for the TimeseriesInput class.
TimeseriesInput.assemble([mapper, ...])	Assembles the HED strings.
TimeseriesInput.column_metadata()	Return the metadata for each column.
TimeseriesInput.combine_dataframe(dataframe)	Combine all columns in the given dataframe into a single HED string series,
TimeseriesInput.convert_to_form(hed_schema, ...)	Convert all tags in underlying dataframe to the specified form.
TimeseriesInput.convert_to_long(hed_schema)	Convert all tags in underlying dataframe to long form.
TimeseriesInput.convert_to_short(hed_schema)	Convert all tags in underlying dataframe to short form.
TimeseriesInput.expand_defs(hed_schema, def_dict)	Shrinks any def-expand found in the underlying dataframe.
TimeseriesInput.get_column_refs()	Return a list of column refs for this file.
TimeseriesInput.get_def_dict(hed_schema[, ...])	Return the definition dict for this file.
TimeseriesInput.get_worksheet([worksheet_name])	Get the requested worksheet.
TimeseriesInput.reset_mapper(new_mapper)	Set mapper to a different view of the file.
TimeseriesInput.set_cell(row_number, ...[, ...])	Replace the specified cell with transformed text.
TimeseriesInput.shrink_defs(hed_schema)	Shrinks any def-expand found in the underlying dataframe.
TimeseriesInput.to_csv([file])	Write to file or return as a string.
TimeseriesInput.to_excel(file)	Output to an Excel file.
TimeseriesInput.validate(hed_schema[, ...])	Creates a SpreadsheetValidator and returns all issues with this file.

Attributes

TimeseriesInput.EXCEL_EXTENSION	
TimeseriesInput.HED_COLUMN_NAME	
TimeseriesInput.TEXT_EXTENSION	
TimeseriesInput.columns	Returns a list of the column names.
TimeseriesInput.dataframe	The underlying dataframe.
TimeseriesInput.dataframe_a	Return the assembled dataframe Probably a placeholder name.
TimeseriesInput.has_column_names	True if dataframe has column names.
TimeseriesInput.loaded_workbook	The underlying loaded workbooks.
TimeseriesInput.name	Name of the data.
TimeseriesInput.needs_sorting	Return True if this both has an onset column, and it needs sorting.
TimeseriesInput.onsets	Return the onset column if it exists.
TimeseriesInput.series_a	Return the assembled dataframe as a series.
TimeseriesInput.series_filtered	Return the assembled dataframe as a series, with rows that have the same onset combined.
TimeseriesInput.worksheet_name	The worksheet name.

TimeseriesInput.**__init__**(*file=None*, *sidecar=None*, *extra_def_dicts=None*, *name=None*)

Constructor for the TimeseriesInput class.

Parameters

- **file** (*str or file like*) – A tsv file to open.
- **sidecar** (*str or Sidecar*) – A json sidecar to pull metadata from.
- **extra_def_dicts** (*DefinitionDict, list, or None*) – Additional definition dictionaries.
- **name** (*str*) – The name to display for this file for error purposes.

Notes

- The extra_def_dicts are external definitions that override the ones in the object.

TimeseriesInput.**assemble**(*mapper=None*, *skip_curly_braces=False*)

Assembles the HED strings.

Parameters

- **mapper** (*ColumnMapper or None*) – Generally pass none here unless you want special behavior.
- **skip_curly_braces** (*bool*) – If True, don't plug in curly brace values into columns.

Returns
The assembled dataframe.

Return type
Dataframe

TimeseriesInput.**column_metadata**()

Return the metadata for each column.

Returns
Number/ColumnMeta pairs.

Return type
dict

static TimeseriesInput.**combine_dataframe**(*dataframe*)

Combine all columns in the given dataframe into a single HED string series,
skipping empty columns and columns with empty strings.

Parameters
dataframe (*Dataframe*) – The dataframe to combine

Returns
The assembled series.

Return type
Series

TimeseriesInput.**convert_to_form**(*hed_schema*, *tag_form*)

Convert all tags in underlying dataframe to the specified form.

Parameters

- **hed_schema** (*HedSchema*) – The schema to use to convert tags.
- **tag_form** (*str*) – HedTag property to convert tags to. Most cases should use convert_to_short or convert_to_long below.

TimeseriesInput.**convert_to_long**(*hed_schema*)

Convert all tags in underlying dataframe to long form.

Parameters
hed_schema (*HedSchema or None*) – The schema to use to convert tags.

TimeseriesInput.**convert_to_short**(*hed_schema*)

Convert all tags in underlying dataframe to short form.

Parameters
hed_schema (*HedSchema*) – The schema to use to convert tags.

TimeseriesInput.**expand_defs**(*hed_schema*, *def_dict*)

Shrinks any def-expand found in the underlying dataframe.

Parameters

- **hed_schema** (*HedSchema or None*) – The schema to use to identify defs.
- **def_dict** (*DefinitionDict*) – The definitions to expand.

TimeseriesInput.**get_column_refs**()

Return a list of column refs for this file.

Default implementation returns none.

Returns
A list of unique column refs found.

Return type
column_refs(list)

TimeseriesInput.**get_def_dict**(*hed_schema*, *extra_def_dicts=None*)

Return the definition dict for this file.

Note: Baseclass implementation returns just extra_def_dicts.

Parameters

- **hed_schema** (*HedSchema*) – Identifies tags to find definitions(if needed).
- **extra_def_dicts** (*list, DefinitionDict, or None*) – Extra dicts to add to the list.

Returns
A single definition dict representing all the data(and extra def dicts).

Return type
DefinitionDict

TimeseriesInput.**get_worksheet**(*worksheet_name=None*)

Get the requested worksheet.

Parameters
worksheet_name (*str or None*) – The name of the requested worksheet by name or the first one if None.

Returns
The workbook request.

Return type
openpyxl.workbook.Workbook

Notes

If None, returns the first worksheet.

Raises
KeyError –

- The specified worksheet name does not exist.

TimeseriesInput.**reset_mapper**(*new_mapper*)

Set mapper to a different view of the file.

Parameters
new_mapper (*ColumnMapper*) – A column mapper to be associated with this base input.

TimeseriesInput.**set_cell**(*row_number*, *column_number*, *new_string_obj*, *tag_form='short_tag'*)

Replace the specified cell with transformed text.

Parameters

- **row_number** (*int*) – The row number of the spreadsheet to set.
- **column_number** (*int*) – The column number of the spreadsheet to set.
- **new_string_obj** (*HedString*) – Object with text to put in the given cell.
- **tag_form** (*str*) – Version of the tags (short_tag, long_tag, base_tag, etc.)

Notes

Any attribute of a HedTag that returns a string is a valid value of tag_form.

Raises

- **ValueError** –
  - There is not a loaded dataframe.
- **KeyError** –
  - The indicated row/column does not exist.
- **AttributeError** –
  - The indicated tag_form is not an attribute of HedTag.

TimeseriesInput.**shrink_defs**(*hed_schema*)

Shrinks any def-expand found in the underlying dataframe.

Parameters
hed_schema (*HedSchema or None*) – The schema to use to identify defs.

TimeseriesInput.**to_csv**(*file=None*)

Write to file or return as a string.

Parameters
file (*str, file-like, or None*) – Location to save this file. If None, return as string.

Returns
None if file is given or the contents as a str if file is None.

Return type
None or str

Raises
OSError –
- Cannot open the indicated file.

TimeseriesInput.**to_excel**(*file*)

Output to an Excel file.

Parameters
file (*str or file-like*) – Location to save this base input.

Raises

- **ValueError** –
  - If empty file object was passed.
- **OSError** –
  - Cannot open the indicated file.

TimeseriesInput.**validate**(*hed_schema*, *extra_def_dicts=None*, *name=None*, *error_handler=None*)

Creates a SpreadsheetValidator and returns all issues with this file.

Parameters

- **hed_schema** (*HedSchema*) – The schema to use for validation.
- **extra_def_dicts** (*list of DefDict or DefDict*) – All definitions to use for validation.

- **name** (*str*) – The name to report errors from this file as.
- **error_handler** (*ErrorHandler*) – Error context to use. Creates a new one if None.

Returns
A list of issues for a HED string.

Return type
issues (list of dict)

TimeseriesInput.**EXCEL_EXTENSION = ['.xlsx']**

TimeseriesInput.**HED_COLUMN_NAME = 'HED'**

TimeseriesInput.**TEXT_EXTENSION = ['.tsv', '.txt']**

TimeseriesInput.**columns**

Returns a list of the column names.

Empty if no column names.

Returns
The column names.

Return type
columns(list)

TimeseriesInput.**dataframe**

The underlying dataframe.

TimeseriesInput.**dataframe_a**

Return the assembled dataframe Probably a placeholder name.

Returns
the assembled dataframe

Return type
Dataframe

TimeseriesInput.**has_column_names**

True if dataframe has column names.

TimeseriesInput.**loaded_workbook**

The underlying loaded workbooks.

TimeseriesInput.**name**

Name of the data.

TimeseriesInput.**needs_sorting**

Return True if this both has an onset column, and it needs sorting.

TimeseriesInput.**onsets**

Return the onset column if it exists.

TimeseriesInput.**series_a**

Return the assembled dataframe as a series.

Returns
the assembled dataframe with columns merged.

Return type
Series

TimeseriesInput.**series_filtered**

Return the assembled dataframe as a series, with rows that have the same onset combined.

Returns
the assembled dataframe with columns merged, and the rows filtered together.

Return type
Series or None

TimeseriesInput.**worksheet_name**

The worksheet name.

3.3 schema

Data structures for handling the HED schema.

Modules

hed.schema.hed_cache	Infrastructure for caching HED schema from remote repositories.
hed.schema.hed_cache_lock	Support utilities for hed_cache locking
hed.schema.hed_schema	
hed.schema.hed_schema_base	Abstract base class for HedSchema and HedSchemaGroup, showing the common functionality
hed.schema.hed_schema_constants	
hed.schema.hed_schema_entry	
hed.schema.hed_schema_group	
hed.schema.hed_schema_io	Utilities for loading and outputting HED schema.
hed.schema.hed_schema_section	
hed.schema.schema_attribute_validator_hed_id	
hed.schema.schema_attribute_validators	The built-in functions to validate known attributes.
hed.schema.schema_compare	Functions supporting comparison of schemas.
hed.schema.schema_compliance	Utilities for HED schema checking.
hed.schema.schema_header_util	
hed.schema.schema_io	
hed.schema.schema_validation_util	Utilities used in HED validation/loading using a HED schema.
hed.schema.schema_validation_util_deprecated	Legacy validation for terms and descriptions prior to 8.3.0.

3.3.1 hed_cache

Infrastructure for caching HED schema from remote repositories.

Functions

cache_local_versions(cache_folder)	Cache all schemas included with the HED installation.
cache_xml_versions([hed_base_urls, ...])	Cache all schemas at the given URLs.
get_cache_directory()	Return the current value of HED_CACHE_DIRECTORY.
get_hed_version_path(xml_version[, ...])	Get HED XML file path in a directory.
get_hed_versions([local_hed_directory, ...])	Get the HED versions in the HED directory.
get_library_data(library_name[, cache_folder])	Retrieve the library data for the given library.
set_cache_directory(new_cache_dir)	Set default global HED cache directory.

cache_local_versions(*cache_folder*)

Cache all schemas included with the HED installation.

Parameters
cache_folder (*str*) – The folder holding the cache.

Returns
Returns -1 on cache access failure. None otherwise

Return type
int or None

cache_xml_versions(*hed_base_urls=('https://api.github.com/repos/hed-standard/hed-schemas/contents/standard_schema',)*, *hed_library_urls=('https://api.github.com/repos/hed-standard/hed-schemas/contents/library_schemas',)*, *skip_folders=('deprecated',)*, *cache_folder=None*)

Cache all schemas at the given URLs.

Parameters

- **hed_base_urls** (*str or list*) – Path or list of paths. These should point to a single folder.
- **hed_library_urls** (*str or list*) – Path or list of paths. These should point to folder containing library folders.
- **skip_folders** (*list*) – A list of subfolders to skip over when downloading.
- **cache_folder** (*str*) – The folder holding the cache.

Returns

Returns -1 if cache failed for any reason, including having been cached too recently.
Returns 0 if it successfully cached this time.

Return type
float

Notes

- The Default skip_folders is 'deprecated'.
- The HED cache folder defaults to HED_CACHE_DIRECTORY.
- **The directories on GitHub are of the form:**
  https://api.github.com/repos/hed-standard/hed-schemas/contents/standard_schema

get_cache_directory()

Return the current value of HED_CACHE_DIRECTORY.

get_hed_version_path(*xml_version*, *library_name=None*, *local_hed_directory=None*, *check_prerelease=False*)

Get HED XML file path in a directory. Only returns filenames that exist.

Parameters

- **library_name** (*str or None*) – Optional the schema library name.
- **xml_version** (*str*) – Returns this version if it exists
- **local_hed_directory** (*str*) – Path to local HED directory. Defaults to HED_CACHE_DIRECTORY
- **check_prerelease** (*bool*) – Also check for prerelease schemas

Returns
The path to the latest HED version the HED directory.

Return type
str

get_hed_versions(*local_hed_directory=None*, *library_name=None*, *check_prerelease=False*)

Get the HED versions in the HED directory.

Parameters

- **local_hed_directory** (*str*) – Directory to check for versions which defaults to hed_cache.
- **library_name** (*str or None*) – An optional schema library name. None retrieves the standard schema only. Pass "all" to retrieve all standard and library schemas as a dict.
- **check_prerelease** (*bool*) – If True, results can include prerelease schemas

Returns
List of version numbers or dictionary {library_name: [versions]}.

Return type
list or dict

get_library_data(*library_name*, *cache_folder=None*)

Retrieve the library data for the given library.

Currently, this is just the valid ID range.

Parameters

- **library_name** (*str*) – The schema name. "" for standard schema.
- **cache_folder** (*str*) – The cache folder to use if not using the default.

Returns
The data for a specific library.

Return type
library_data(dict)

set_cache_directory(*new_cache_dir*)

Set default global HED cache directory.

Parameters
new_cache_dir (*str*) – Directory to check for versions.

3.3.2 hed_cache_lock

Support utilities for hed_cache locking

Classes

CacheLock(cache_folder[, write_time, ...])	Class to lock the cache folder to ensure it doesn't get hit by another version at the same time.

3.3.2.1 CacheLock

class CacheLock(*cache_folder*, *write_time=True*, *time_threshold=1800*)

Class to lock the cache folder to ensure it doesn't get hit by another version at the same time.

Methods

CacheLock.__init__(cache_folder[, ...])	Constructor for hed locking object

Attributes

CacheLock.**__init__**(*cache_folder*, *write_time=True*, *time_threshold=1800*)

Constructor for hed locking object

Parameters

- **cache_folder** (*str*) – The folder to create the lock in(implicitly locking that folder)
- **write_time** (*bool*) – If true, read and write the cache time. Additionally, won't operate if too recent. Generally False for local operations.
- **time_threshold** (*int*) – Time before cache is allowed to refresh again.

Exceptions

CacheException	Exception for cache locking or threshold errors.

3.3.2.2 hed.schema.hed_cache_lock.CacheException

exception CacheException

Exception for cache locking or threshold errors.

3.3.3 hed_schema

Classes

HedSchema()	A HED schema suitable for processing.

3.3.3.1 HedSchema

class HedSchema

A HED schema suitable for processing.

Methods

HedSchema.__init__()	Constructor for the HedSchema class.
HedSchema.can_save()	Returns if it's legal to save this schema.
HedSchema.check_compliance([...])	Check for HED3 compliance of this schema.
HedSchema.finalize_dictionaries()	Call to finish loading.
HedSchema.find_tag_entry(tag[, schema_namespace])	Find the schema entry for a given source tag.
HedSchema.get_as_dataframes([save_merged])	Get a dict of dataframes representing this file
HedSchema.get_as_mediawiki_string([save_merged])	Return the schema to a mediawiki string.
HedSchema.get_as_xml_string([save_merged])	Return the schema to an XML string.
HedSchema.get_extras(extras_key)	Get the extras corresponding to the given key
HedSchema.get_formatted_version()	The HED version string including namespace and library name if any of this schema.
HedSchema.get_save_header_attributes([...])	returns the attributes that should be saved.
HedSchema.get_schema_versions()	A list of HED version strings including namespace and library name if any of this schema.
HedSchema.get_tag_attribute_names_old()	Return a dict of all allowed tag attributes.
HedSchema.get_tag_entry(name[, key_class, ...])	Return the schema entry for this tag, if one exists.
HedSchema.get_tags_with_attribute(attribute)	Return tag entries with the given attribute.
HedSchema.has_duplicates()	Returns the first duplicate tag/unit/etc.
HedSchema.save_as_dataframes(base_filename)	Save as dataframes to a folder of files.
HedSchema.save_as_mediawiki(filename[, ...])	Save as mediawiki to a file.
HedSchema.save_as_xml(filename[, save_merged])	Save as XML to a file.
HedSchema.schema_for_namespace(namespace)	Return HedSchema object for this namespace.
HedSchema.set_schema_prefix(schema_namespace)	Set library namespace associated for this schema.

Attributes

HedSchema.attributes	Return the attributes schema section.
HedSchema.library	The name of this library schema if one exists.
HedSchema.merged	Returns if this schema was loaded from a merged file
HedSchema.name	User provided name for this schema, defaults to filename or version if no name provided.
HedSchema.properties	Return the properties schema section.
HedSchema.schema_83_props	Returns if this is an 8.3.0 or greater schema.
HedSchema.schema_namespace	Returns the schema namespace prefix
HedSchema.tags	Return the tag schema section.
HedSchema.unit_classes	Return the unit classes schema section.
HedSchema.unit_modifiers	Return the modifiers classes schema section
HedSchema.units	Return the unit schema section.
HedSchema.valid_prefixes	Return a list of all prefixes this schema will accept
HedSchema.value_classes	Return the value classes schema section.
HedSchema.version	The complete schema version, including prefix and library name(if applicable)
HedSchema.version_number	The HED version of this schema.
HedSchema.with_standard	The version of the base schema this is extended from, if it exists.

HedSchema.**__init__**()

Constructor for the HedSchema class.

A HedSchema can be used for validation, checking tag attributes, parsing tags, etc.

HedSchema.**can_save**()

Returns if it's legal to save this schema.

You cannot save schemas loaded as merged from multiple library schemas.

Returns
True if this can be saved

Return type
bool

HedSchema.**check_compliance**(*check_for_warnings=True*, *name=None*, *error_handler=None*)

Check for HED3 compliance of this schema.

Parameters

- **check_for_warnings** (*bool*) – If True, checks for formatting issues like invalid characters, capitalization.
- **name** (*str*) – If present, use as the filename for context, rather than using the actual filename. Useful for temp filenames when supporting web services.
- **error_handler** (*ErrorHandler or None*) – Used to report errors. Uses a default one if none passed in.

Returns
A list of all warnings and errors found in the file. Each issue is a dictionary.

Return type
list

HedSchema.**finalize_dictionaries**()

Call to finish loading.

HedSchema.**find_tag_entry**(*tag*, *schema_namespace=''*)

Find the schema entry for a given source tag.

Parameters

- **tag** (*str, HedTag*) – Any form of tag to look up. Can have an extension, value, etc.
- **schema_namespace** (*str*) – The schema namespace of the tag, if any.

Returns

The located tag entry for this tag. str: The remainder of the tag that isn't part of the base tag. list: A list of errors while converting.

Return type

HedTagEntry

Notes

Works left to right (which is mostly relevant for errors).

HedSchema.**get_as_dataframes**(*save_merged=False*)

Get a dict of dataframes representing this file

Parameters

save_merged (*bool*) – If True, returns DFs as if merged with standard.

Returns

a dict of dataframes you can load as a schema

Return type

dataframes(dict)

HedSchema.**get_as_mediawiki_string**(*save_merged=False*)

Return the schema to a mediawiki string.

Parameters

save_merged (*bool*) – If True, this will save the schema as a merged schema if it is a "withStandard" schema. If it is not a "withStandard" schema, this setting has no effect.

Returns

The schema as a string in mediawiki format.

Return type

str

HedSchema.**get_as_xml_string**(*save_merged=True*)

Return the schema to an XML string.

Parameters

- **save_merged** (*bool*) –
- **True** (*If*) –
- **schema.** (*this will save the schema as a merged schema if it is a "withStandard"*) –
- **schema** (*If it is not a "withStandard"*) –
- **effect.** (*this setting has no*) –

Returns
Return the schema as an XML string.

Return type
str

HedSchema.**get_extras**(*extras_key*)

Get the extras corresponding to the given key

Parameters
extras_key (*str*) – The key to check for in the extras dictionary.

Returns
True if the extras dictionary has this key.

Return type
DataFrame

HedSchema.**get_formatted_version**()

The HED version string including namespace and library name if any of this schema.

Returns
A json formatted string of the complete version of this schema including library name and namespace.

Return type
str

HedSchema.**get_save_header_attributes**(*save_merged=False*)

returns the attributes that should be saved.

HedSchema.**get_schema_versions**()

A list of HED version strings including namespace and library name if any of this schema.

Returns
The complete version of this schema including library name and namespace.

Return type
list

HedSchema.**get_tag_attribute_names_old**()

Return a dict of all allowed tag attributes.

Returns
A dictionary whose keys are attribute names and values are HedSchemaEntry object.

Return type
dict

HedSchema.**get_tag_entry**(*name*, *key_class=HedSectionKey.Tags*, *schema_namespace=''*)

Return the schema entry for this tag, if one exists.

Parameters

- **name** (*str*) – Any form of basic tag(or other section entry) to look up. This will not handle extensions or similar. If this is a tag, it can have a schema namespace, but it's not required
- **key_class** (*HedSectionKey or str*) – The type of entry to return.
- **schema_namespace** (*str*) – Only used on Tags. If incorrect, will return None.

Returns
The schema entry for the given tag.

Return type
HedSchemaEntry

HedSchema.**get_tags_with_attribute**(*attribute*, *key_class=HedSectionKey.Tags*)

Return tag entries with the given attribute.

Parameters

- **attribute** (*str*) – A tag attribute. Eg HedKey.ExtensionAllowed
- **key_class** (*HedSectionKey*) – The HedSectionKey for the section to retrieve from.

Returns
A list of all tags with this attribute.

Return type
list

Notes

- The result is cached so will be fast after first call.

HedSchema.**has_duplicates**()

Returns the first duplicate tag/unit/etc. if any section has a duplicate name

HedSchema.**save_as_dataframes**(*base_filename*, *save_merged=False*)

Save as dataframes to a folder of files.

If base_filename has a .tsv suffix, save directly to the indicated location. If base_filename is a directory(does NOT have a .tsv suffix), save the contents into a directory named that. The subfiles are named the same. e.g. HED8.3.0/HED8.3.0_Tag.tsv

base_filename: str
save filename. A suffix will be added to most, e.g. _Tag

save_merged: bool
If True, this will save the schema as a merged schema if it is a "withStandard" schema. If it is not a "withStandard" schema, this setting has no effect.

Raises
OSError –

- File cannot be saved for some reason.

HedSchema.**save_as_mediawiki**(*filename*, *save_merged=False*)

Save as mediawiki to a file.

filename: str
save location

save_merged: bool
If True, this will save the schema as a merged schema if it is a "withStandard" schema. If it is not a "withStandard" schema, this setting has no effect.

Raises
OSError –

- File cannot be saved for some reason.

HedSchema.**save_as_xml**(*filename*, *save_merged=True*)

Save as XML to a file.

filename: str
save location

save_merged: bool
If true, this will save the schema as a merged schema if it is a "withStandard" schema. If it is not a "withStandard" schema, this setting has no effect.

Raises
OSError –

- File cannot be saved for some reason

HedSchema.**schema_for_namespace**(*namespace*)

Return HedSchema object for this namespace.

Parameters
namespace (*str*) – The schema library name namespace.

Returns
The HED schema object for this schema.

Return type
HedSchema

HedSchema.**set_schema_prefix**(*schema_namespace*)

Set library namespace associated for this schema.

Parameters
schema_namespace (*str*) – Should be empty, or end with a colon.(Colon will be automated added if missing).

Raises
HedFileError –

- The prefix is invalid

HedSchema.**attributes**

Return the attributes schema section.

Returns
The attributes section.

Return type
HedSchemaSection

HedSchema.**library**

The name of this library schema if one exists.

Returns
Library name if any.

Return type
str

HedSchema.**merged**

Returns if this schema was loaded from a merged file

Returns
True if file was loaded from a merged file

Return type
bool

HedSchema.**name**

User provided name for this schema, defaults to filename or version if no name provided.

HedSchema.**properties**

Return the properties schema section.

Returns
The properties section.

Return type
HedSchemaSection

HedSchema.**schema_83_props**

Returns if this is an 8.3.0 or greater schema.

Returns
True if standard or partnered schema is 8.3.0 or greater.

Return type
is_83_schema(bool)

HedSchema.**schema_namespace**

Returns the schema namespace prefix

HedSchema.**tags**

Return the tag schema section.

Returns
The tag section.

Return type
HedSchemaTagSection

HedSchema.**unit_classes**

Return the unit classes schema section.

Returns
The unit classes section.

Return type
HedSchemaUnitClassSection

HedSchema.**unit_modifiers**

Return the modifiers classes schema section

Returns
The unit modifiers section.

Return type
HedSchemaSection

HedSchema.**units**

Return the unit schema section.

Returns
The unit section.

Return type
HedSchemaSection

HedSchema.**valid_prefixes**

Return a list of all prefixes this schema will accept

Returns
A list of valid tag prefixes for this schema.

Return type
list

Notes

- The return value is always length 1 if using a HedSchema.

HedSchema.**value_classes**

Return the value classes schema section.

Returns
The value classes section.

Return type
HedSchemaSection

HedSchema.**version**

The complete schema version, including prefix and library name(if applicable)

HedSchema.**version_number**

The HED version of this schema.

Returns
The version of this schema.

Return type
str

HedSchema.**with_standard**

The version of the base schema this is extended from, if it exists.

Returns
HED version or “”

Return type
str

3.3.4 hed_schema_base

Abstract base class for HedSchema and HedSchemaGroup, showing the common functionality

Classes

HedSchemaBase()	Baseclass for schema and schema group.

3.3.4.1 HedSchemaBase

class HedSchemaBase

Baseclass for schema and schema group.

Implementing the abstract functions will allow you to use the schema for validation

Methods

HedSchemaBase.__init__()	

HedSchemaBase.check_compliance([...])	Check for HED3 compliance of this schema.
HedSchemaBase.find_tag_entry(tag[, ...])	Find the schema entry for a given source tag.
HedSchemaBase.get_formatted_version()	The HED version string including namespace and library name if any of this schema.
HedSchemaBase.get_schema_versions()	A list of HED version strings including namespace and library name if any of this schema.
HedSchemaBase.get_tag_entry(name[, ...])	Return the schema entry for this tag, if one exists.
HedSchemaBase.get_tags_with_attribute(attribute)	Return tag entries with the given attribute.
HedSchemaBase.schema_for_namespace(namespace)	Return the HedSchema for the library namespace.

Attributes

HedSchemaBase.name	User provided name for this schema, defaults to filename or version if no name provided.
HedSchemaBase.schema_83_props	Returns if this is an 8.3.0 or greater schema.
HedSchemaBase.valid_prefixes	Return a list of all prefixes this group will accept.

HedSchemaBase.**__init__**()

abstract HedSchemaBase.**check_compliance**(*check_for_warnings=True*, *name=None*, *error_handler=None*)

Check for HED3 compliance of this schema.

Parameters

- **check_for_warnings** (*bool*) – If True, checks for formatting issues like invalid characters, capitalization.
- **name** (*str*) – If present, use as the filename for context, rather than using the actual filename. Useful for temp filenames when supporting web services.
- **error_handler** (*ErrorHandler or None*) – Used to report errors. Uses a default one if none passed in.

Returns

A list of all warnings and errors found in the file. Each issue is a dictionary.

Return type
list

abstract HedSchemaBase.**find_tag_entry**(*tag*, *schema_namespace=''*)

Find the schema entry for a given source tag.

Parameters

- **tag** (*str, HedTag*) – Any form of tag to look up. Can have an extension, value, etc.
- **schema_namespace** (*str*) – The schema namespace of the tag, if any.

Returns
The located tag entry for this tag. str: The remainder of the tag that isn't part of the base tag. list: A list of errors while converting.

Return type
HedTagEntry

Notes

Works left to right (which is mostly relevant for errors).

abstract HedSchemaBase.**get_formatted_version**()

The HED version string including namespace and library name if any of this schema.

Returns
The complete version of this schema including library name and namespace.

Return type
str

abstract HedSchemaBase.**get_schema_versions**()

A list of HED version strings including namespace and library name if any of this schema.

Returns
The complete version of this schema including library name and namespace.

Return type
list

abstract HedSchemaBase.**get_tag_entry**(*name*, *key_class=HedSectionKey.Tags*, *schema_namespace=''*)

Return the schema entry for this tag, if one exists.

Parameters

- **name** (*str*) – Any form of basic tag(or other section entry) to look up. This will not handle extensions or similar. If this is a tag, it can have a schema namespace, but it's not required
- **key_class** (*HedSectionKey or str*) – The type of entry to return.
- **schema_namespace** (*str*) – Only used on Tags. If incorrect, will return None.

Returns
The schema entry for the given tag.

Return type
HedSchemaEntry

abstract HedSchemaBase.**get_tags_with_attribute**(*attribute*, *key_class=HedSectionKey.Tags*)

Return tag entries with the given attribute.

Parameters

- **attribute** (*str*) – A tag attribute. Eg HedKey.ExtensionAllowed
- **key_class** (*HedSectionKey*) – The HedSectionKey for the section to retrieve from.

Returns
A list of all tags with this attribute.

Return type
list

Notes

- The result is cached so will be fast after first call.

abstract HedSchemaBase.**schema_for_namespace**(*namespace*)

Return the HedSchema for the library namespace.

Parameters
namespace (*str*) – A schema library name namespace.

Returns
The specific schema for this library name namespace if exists.

Return type
HedSchema or None

HedSchemaBase.**name**

User provided name for this schema, defaults to filename or version if no name provided.

HedSchemaBase.**schema_83_props**

Returns if this is an 8.3.0 or greater schema.

Returns
True if standard or partnered schema is 8.3.0 or greater.

Return type
is_83_schema(bool)

HedSchemaBase.**valid_prefixes**

Return a list of all prefixes this group will accept.

Returns
A list of strings representing valid prefixes for this group.

Return type
prefixes(list of str)

3.3.5 hed_schema_constants

Classes

HedKey()	Known property and attribute names.
HedKeyOld()	
HedSectionKey(value)	Keys designating specific sections in a HedSchema object.

3.3.5.1 HedKey

class HedKey

Known property and attribute names.

Notes

- These names should match the attribute values in the XML/wiki.

Methods

HedKey.__init__()

Attributes

HedKey.AllowedCharacter
HedKey.AnnotationProperty
HedKey.BoolRange
HedKey.ConversionFactor
HedKey.DefaultUnits
HedKey.DeprecatedFrom
HedKey.ElementDomain
HedKey.ExtensionAllowed
HedKey.HedID
HedKey.InLibrary

continues on next page

Table 5 – continued from previous page

HedKey.NumericRange
HedKey.Recommended
HedKey.RelatedTag
HedKey.RequireChild
HedKey.Required
HedKey.Reserved
HedKey.Rooted
HedKey.SIUnit
HedKey.SIUnitModifier
HedKey.SIUnitSymbolModifier
HedKey.StringRange
HedKey.SuggestedTag
HedKey.TagDomain
HedKey.TagGroup
HedKey.TagRange
HedKey.TakesValue
HedKey.TopLevelTagGroup
HedKey.Unique
HedKey.UnitClass
HedKey.UnitClassDomain
HedKey.UnitClassRange
HedKey.UnitDomain
HedKey.UnitModifierDomain
HedKey.UnitPrefix
HedKey.UnitRange

continues on next page

Table 5 – continued from previous page

HedKey.UnitSymbol
HedKey.ValueClass
HedKey.ValueClassDomain
HedKey.ValueClassRange

HedKey.__init__()

HedKey.AllowedCharacter = 'allowedCharacter'

HedKey.AnnotationProperty = 'annotationProperty'

HedKey.BoolRange = 'boolRange'

HedKey.ConversionFactor = 'conversionFactor'

HedKey.DefaultUnits = 'defaultUnits'

HedKey.DeprecatedFrom = 'deprecatedFrom'

HedKey.ElementDomain = 'elementDomain'

HedKey.ExtensionAllowed = 'extensionAllowed'

HedKey.HedID = 'hedId'

HedKey.InLibrary = 'inLibrary'

HedKey.NumericRange = 'numericRange'

HedKey.Recommended = 'recommended'

HedKey.RelatedTag = 'relatedTag'

HedKey.RequireChild = 'requireChild'

HedKey.Required = 'required'

HedKey.Reserved = 'reserved'

HedKey.Rooted = 'rooted'

HedKey.SIUnit = 'SIUnit'

HedKey.SIUnitModifier = 'SIUnitModifier'

HedKey.SIUnitSymbolModifier = 'SIUnitSymbolModifier'

HedKey.StringRange = 'stringRange'

HedKey.SuggestedTag = 'suggestedTag'

HedKey.TagDomain = 'tagDomain'

HedKey.TagGroup = 'tagGroup'

HedKey.TagRange = 'tagRange'

HedKey.TakesValue = 'takesValue'

HedKey.TopLevelTagGroup = 'topLevelTagGroup'

HedKey.Unique = 'unique'

HedKey.UnitClass = 'unitClass'

HedKey.UnitClassDomain = 'unitClassDomain'

HedKey.UnitClassRange = 'unitClassRange'

HedKey.UnitDomain = 'unitDomain'

HedKey.UnitModifierDomain = 'unitModifierDomain'

HedKey.UnitPrefix = 'unitPrefix'

HedKey.UnitRange = 'unitRange'

HedKey.UnitSymbol = 'unitSymbol'

HedKey.ValueClass = 'valueClass'

HedKey.ValueClassDomain = 'valueClassDomain'

HedKey.ValueClassRange = 'valueClassRange'

3.3.5.2 HedKeyOld

class HedKeyOld

Methods

HedKeyOld.__init__()

Attributes

HedKeyOld.BoolProperty
HedKeyOld.ElementProperty
HedKeyOld.IsInheritedProperty
HedKeyOld.NodeProperty
HedKeyOld.UnitClassProperty
HedKeyOld.UnitModifierProperty
HedKeyOld.UnitProperty
HedKeyOld.ValueClassProperty

HedKeyOld.**__init__**()

HedKeyOld.**BoolProperty** = **'boolProperty'**

HedKeyOld.**ElementProperty** = **'elementProperty'**

HedKeyOld.**IsInheritedProperty** = **'isInheritedProperty'**

HedKeyOld.**NodeProperty** = **'nodeProperty'**

HedKeyOld.**UnitClassProperty** = **'unitClassProperty'**

HedKeyOld.**UnitModifierProperty** = **'unitModifierProperty'**

HedKeyOld.**UnitProperty** = **'unitProperty'**

HedKeyOld.**ValueClassProperty** = **'valueClassProperty'**

3.3.5.3 HedSectionKey

class HedSectionKey(*value*)

Keys designating specific sections in a HedSchema object.

Methods

Attributes

HedSectionKey.Tags
HedSectionKey.UnitClasses
HedSectionKey.Units
HedSectionKey.UnitModifiers
HedSectionKey.ValueClasses
HedSectionKey.Attributes
HedSectionKey.Properties

HedSectionKey.**Tags** = **'tags'**

HedSectionKey.**UnitClasses** = **'unitClasses'**

HedSectionKey.**Units** = **'units'**

HedSectionKey.**UnitModifiers** = **'unitModifiers'**

HedSectionKey.**ValueClasses** = **'valueClasses'**

HedSectionKey.**Attributes** = **'attributes'**

HedSectionKey.**Properties** = **'properties'**

3.3.6 hed_schema_entry

Classes

HedSchemaEntry(name, section)	A single node in a HedSchema.
HedTagEntry(*args, **kwargs)	A single tag entry in the HedSchema.
UnitClassEntry(*args, **kwargs)	A single unit class entry in the HedSchema.
UnitEntry(*args, **kwargs)	A single unit entry with modifiers in the HedSchema.

3.3.6.1 HedSchemaEntry

class HedSchemaEntry(*name*, *section*)

A single node in a HedSchema.

The structure contains all the node information including attributes and properties.

Methods

HedSchemaEntry.__init__(name, section)	Constructor for HedSchemaEntry.
HedSchemaEntry.attribute_has_property(...)	Return True if attribute has property.
HedSchemaEntry.finalize_entry(schema)	Called once after loading to set internal state.
HedSchemaEntry.has_attribute(attribute[, ...])	Checks for the existence of an attribute in this entry.

Attributes

HedSchemaEntry.section_key

HedSchemaEntry.**__init__**(*name*, *section*)

Constructor for HedSchemaEntry.

Parameters

- **name** (*str*) – The name of the entry.
- **section** (*HedSchemaSection*) – The section to which it belongs.

HedSchemaEntry.**attribute_has_property**(*attribute*, *property_name*)

Return True if attribute has property.

Parameters

- **attribute** (*str*) – Attribute name to check for property_name.
- **property_name** (*str*) – The property value to return.

Returns
Returns True if this entry has the property.

Return type
bool

HedSchemaEntry.**finalize_entry**(*schema*)

Called once after loading to set internal state.

Parameters
schema (*HedSchema*) – The schema that holds the rules.

HedSchemaEntry.**has_attribute**(*attribute*, *return_value=False*)

Checks for the existence of an attribute in this entry.

Parameters

- **attribute** (*str*) – The attribute to check for.
- **return_value** (*bool*) – If True, returns the actual value of the attribute. If False, returns a boolean indicating the presence of the attribute.

Returns
If return_value is False, returns True if the attribute exists and False otherwise. If return_value is True, returns the value of the attribute if it exists, else returns None.

Return type
bool or any

Notes

- The existence of an attribute does not guarantee its validity.

HedSchemaEntry.**section_key**

3.3.6.2 HedTagEntry

class HedTagEntry(**args*, ***kwargs*)

A single tag entry in the HedSchema.

Methods

HedTagEntry.__init__(*args, **kwargs)	Constructor for HedSchemaEntry.
HedTagEntry.attribute_has_property(...)	Return True if attribute has property.
HedTagEntry.base_tag_has_attribute(tag_attribute)	Check if the base tag has a specific attribute.
HedTagEntry.finalize_entry(schema)	Called once after schema loading to set state.
HedTagEntry.has_attribute(attribute[, ...])	Returns th existence or value of an attribute in this entry.

Attributes

HedTagEntry.parent	Get the parent entry of this tag
HedTagEntry.parent_name	Gets the parent tag entry name
HedTagEntry.section_key	

HedTagEntry.**__init__**(**args*, ***kwargs*)

Constructor for HedSchemaEntry.

Parameters

- **name** (*str*) – The name of the entry.
- **section** (*HedSchemaSection*) – The section to which it belongs.

HedTagEntry.**attribute_has_property**(*attribute*, *property_name*)

Return True if attribute has property.

Parameters

- **attribute** (*str*) – Attribute name to check for property_name.
- **property_name** (*str*) – The property value to return.

Returns

Returns True if this entry has the property.

Return type

bool

HedTagEntry.**base_tag_has_attribute**(*tag_attribute*)

Check if the base tag has a specific attribute.

Parameters
tag_attribute (*str*) – A tag attribute.

Returns
True if the tag has the specified attribute. False, if otherwise.

Return type
bool

Notes

This mostly is relevant for takes value tags.

HedTagEntry.**finalize_entry**(*schema*)

Called once after schema loading to set state.

Parameters
schema (*HedSchema*) – The schema that the rules come from.

HedTagEntry.**has_attribute**(*attribute*, *return_value=False*)

Returns th existence or value of an attribute in this entry.

This also checks parent tags for inheritable attributes like ExtensionAllowed.

Parameters

- **attribute** (*str*) – The attribute to check for.
- **return_value** (*bool*) – If True, returns the actual value of the attribute. If False, returns a boolean indicating the presence of the attribute.

Returns
If return_value is False, returns True if the attribute exists and False otherwise. If return_value is True, returns the value of the attribute if it exists, else returns None.

Return type
bool or any

Notes

- The existence of an attribute does not guarantee its validity.

HedTagEntry.**parent**

Get the parent entry of this tag

HedTagEntry.**parent_name**

Gets the parent tag entry name

HedTagEntry.**section_key**

3.3.6.3 UnitClassEntry

class UnitClassEntry(**args*, ***kwargs*)

A single unit class entry in the HedSchema.

Methods

UnitClassEntry.__init__(*args, **kwargs)	Constructor for HedSchemaEntry.
UnitClassEntry.add_unit(unit_entry)	Add the given unit entry to this unit class.
UnitClassEntry.attribute_has_property(...)	Return True if attribute has property.
UnitClassEntry.finalize_entry(schema)	Called once after schema load to set state.
UnitClassEntry.get_derivative_unit_entry(units)	Gets the (derivative) unit entry if it exists
UnitClassEntry.has_attribute(attribute[, ...])	Checks for the existence of an attribute in this entry.

Attributes

UnitClassEntry.children	Alias to get the units for this class
UnitClassEntry.section_key	

UnitClassEntry.**__init__**(**args*, ***kwargs*)

Constructor for HedSchemaEntry.

Parameters

- **name** (*str*) – The name of the entry.
- **section** (*HedSchemaSection*) – The section to which it belongs.

UnitClassEntry.**add_unit**(*unit_entry*)

Add the given unit entry to this unit class.

Parameters

unit_entry (*HedSchemaEntry*) – Unit entry to add.

UnitClassEntry.**attribute_has_property**(*attribute*, *property_name*)

Return True if attribute has property.

Parameters

- **attribute** (*str*) – Attribute name to check for property_name.
- **property_name** (*str*) – The property value to return.

Returns

Returns True if this entry has the property.

Return type

bool

UnitClassEntry.**finalize_entry**(*schema*)

Called once after schema load to set state.

Parameters

schema (*HedSchema*) – The object with the schema rules.

UnitClassEntry.**get_derivative_unit_entry**(*units*)

Gets the (derivative) unit entry if it exists

Parameters
units (*str*) – The unit name to check, can be plural or include a modifier.

Returns
The unit entry if it exists

Return type
unit_entry(UnitEntry or None)

UnitClassEntry.**has_attribute**(*attribute*, *return_value=False*)

Checks for the existence of an attribute in this entry.

Parameters

- **attribute** (*str*) – The attribute to check for.
- **return_value** (*bool*) – If True, returns the actual value of the attribute. If False, returns a boolean indicating the presence of the attribute.

Returns
If return_value is False, returns True if the attribute exists and False otherwise. If return_value is True, returns the value of the attribute if it exists, else returns None.

Return type
bool or any

Notes

- The existence of an attribute does not guarantee its validity.

UnitClassEntry.**children**

Alias to get the units for this class

Returns
The unit list for this class

Return type
unit_list(list)

UnitClassEntry.**section_key**

3.3.6.4 UnitEntry

class UnitEntry(**args*, ***kwargs*)

A single unit entry with modifiers in the HedSchema.

Methods

UnitEntry.__init__(*args, **kwargs)	Constructor for HedSchemaEntry.
UnitEntry.attribute_has_property(attribute, ...)	Return True if attribute has property.
UnitEntry.finalize_entry(schema)	Called once after loading to set internal state.
UnitEntry.get_conversion_factor(unit_name)	Returns the conversion factor from combining this unit with the specified modifier
UnitEntry.has_attribute(attribute[, ...])	Checks for the existence of an attribute in this entry.

Attributes

UnitEntry.section_key

UnitEntry.**__init__**(**args*, ***kwargs*)

Constructor for HedSchemaEntry.

Parameters

- **name** (*str*) – The name of the entry.
- **section** (*HedSchemaSection*) – The section to which it belongs.

UnitEntry.**attribute_has_property**(*attribute*, *property_name*)

Return True if attribute has property.

Parameters

- **attribute** (*str*) – Attribute name to check for property_name.
- **property_name** (*str*) – The property value to return.

Returns

Returns True if this entry has the property.

Return type

bool

UnitEntry.**finalize_entry**(*schema*)

Called once after loading to set internal state.

Parameters

schema (*HedSchema*) – The schema rules come from.

UnitEntry.**get_conversion_factor**(*unit_name*)

Returns the conversion factor from combining this unit with the specified modifier

Parameters

unit_name (*str or None*) – the full name of the unit with modifier

Returns

Returns the conversion factor or None

Return type

conversion_factor(float or None)

UnitEntry.**has_attribute**(*attribute*, *return_value=False*)

Checks for the existence of an attribute in this entry.

Parameters

- **attribute** (*str*) – The attribute to check for.
- **return_value** (*bool*) – If True, returns the actual value of the attribute. If False, returns a boolean indicating the presence of the attribute.

Returns

If return_value is False, returns True if the attribute exists and False otherwise. If return_value is True, returns the value of the attribute if it exists, else returns None.

Return type

bool or any

Notes

- The existence of an attribute does not guarantee its validity.

UnitEntry.**section_key**

3.3.7 hed_schema_group

Classes

HedSchemaGroup(schema_list[, name])	Container for multiple HedSchema objects.

3.3.7.1 HedSchemaGroup

class HedSchemaGroup(*schema_list*, *name=''*)

Container for multiple HedSchema objects.

Notes

- The container class is useful when library schema are included.
- You cannot save/load/etc. the combined schema object directly.

Methods

HedSchemaGroup.__init__(schema_list[, name])	Combine multiple HedSchema objects from a list.
HedSchemaGroup.check_compliance([...])	Check for HED3 compliance of this schema.
HedSchemaGroup.find_tag_entry(tag[, ...])	Find the schema entry for a given source tag.
HedSchemaGroup.get_formatted_version()	The HED version string including namespace and library name if any of this schema.
HedSchemaGroup.get_schema_versions()	A list of HED version strings including namespace and library name if any of this schema.
HedSchemaGroup.get_tag_entry(name[, ...])	Return the schema entry for this tag, if one exists.
HedSchemaGroup.get_tags_with_attribute(attribute)	Return tag entries with the given attribute.
HedSchemaGroup.schema_for_namespace(namespace)	Return the HedSchema for the library namespace.

Attributes

HedSchemaGroup.name	User provided name for this schema, defaults to filename or version if no name provided.
HedSchemaGroup.schema_83_props	Returns if this is an 8.3.0 or greater schema.
HedSchemaGroup.valid_prefixes	Return a list of all prefixes this group will accept.

HedSchemaGroup.**__init__**(*schema_list*, *name=''*)

Combine multiple HedSchema objects from a list.

Parameters

schema_list (*list*) – A list of HedSchema for the container.

Returns

the container created.

Return type

HedSchemaGroup

Raises

HedFileError –

- Multiple schemas have the same library prefixes.
- Empty list passed

HedSchemaGroup.**check_compliance**(*check_for_warnings=True*, *name=None*, *error_handler=None*)

Check for HED3 compliance of this schema.

Parameters

- **check_for_warnings** (*bool*) – If True, checks for formatting issues like invalid characters, capitalization.
- **name** (*str*) – If present, use as the filename for context, rather than using the actual filename. Useful for temp filenames when supporting web services.
- **error_handler** (*ErrorHandler or None*) – Used to report errors. Uses a default one if none passed in.

Returns

A list of all warnings and errors found in the file. Each issue is a dictionary.

Return type
list

HedSchemaGroup.**find_tag_entry**(*tag*, *schema_namespace=''*)

Find the schema entry for a given source tag.

Parameters

- **tag** (*str, HedTag*) – Any form of tag to look up. Can have an extension, value, etc.
- **schema_namespace** (*str*) – The schema namespace of the tag, if any.

Returns
The located tag entry for this tag. str: The remainder of the tag that isn't part of the base tag. list: A list of errors while converting.

Return type
HedTagEntry

Notes

Works left to right (which is mostly relevant for errors).

HedSchemaGroup.**get_formatted_version**()

The HED version string including namespace and library name if any of this schema.

Returns
The complete version of this schema including library name and namespace.

Return type
str

HedSchemaGroup.**get_schema_versions**()

A list of HED version strings including namespace and library name if any of this schema.

Returns
The complete version of this schema including library name and namespace.

Return type
list

HedSchemaGroup.**get_tag_entry**(*name*, *key_class=HedSectionKey.Tags*, *schema_namespace=''*)

Return the schema entry for this tag, if one exists.

Parameters

- **name** (*str*) – Any form of basic tag(or other section entry) to look up. This will not handle extensions or similar. If this is a tag, it can have a schema namespace, but it's not required
- **key_class** (*HedSectionKey or str*) – The type of entry to return.
- **schema_namespace** (*str*) – Only used on Tags. If incorrect, will return None.

Returns
The schema entry for the given tag.

Return type
HedSchemaEntry

HedSchemaGroup.**get_tags_with_attribute**(*attribute*, *key_class=HedSectionKey.Tags*)

Return tag entries with the given attribute.

Parameters

- **attribute** (*str*) – A tag attribute. Eg HedKey.ExtensionAllowed
- **key_class** (*HedSectionKey*) – The HedSectionKey for the section to retrieve from.

Returns
A list of all tags with this attribute.

Return type
list

Notes

- The result is cached so will be fast after first call.

HedSchemaGroup.**schema_for_namespace**(*namespace*)

Return the HedSchema for the library namespace.

Parameters
namespace (*str*) – A schema library name namespace.

Returns
The specific schema for this library name namespace if exists.

Return type
HedSchema or None

HedSchemaGroup.**name**

User provided name for this schema, defaults to filename or version if no name provided.

HedSchemaGroup.**schema_83_props**

Returns if this is an 8.3.0 or greater schema.

Returns
True if standard or partnered schema is 8.3.0 or greater.

Return type
is_83_schema(bool)

HedSchemaGroup.**valid_prefixes**

Return a list of all prefixes this group will accept.

Returns
A list of strings representing valid prefixes for this group.

Return type
list

3.3.8 hed_schema_io

Utilities for loading and outputting HED schema.

Functions

from_dataframes(schema_data[, ...])	Create a schema from the given string.
from_string(schema_string[, schema_format, ...])	Create a schema from the given string.
get_hed_xml_version(xml_file_path)	Get the version number from a HED XML file.
load_schema(hed_path[, schema_namespace, ...])	Load a schema from the given file or URL path.
load_schema_version([xml_version, xml_folder])	Return a HedSchema or HedSchemaGroup extracted from xml_version
parse_version_list(xml_version_list)	Takes a list of xml versions and returns a dictionary split by prefix

from_dataframes(*schema_data*, *schema_namespace=None*, *name=None*)

Create a schema from the given string.

Parameters

- **schema_string** (*dict*) – A dict of DF_SUFFIXES:file_as_string_or_df Should have an entry for all values of DF_SUFFIXES.
- **schema_namespace** (*str, None*) – The name_prefix all tags in this schema will accept.
- **name** (*str or None*) – User supplied identifier for this schema

Returns
The loaded schema.

Return type
(HedSchema)

Raises
HedFileError –

- Empty/invalid parameters

Notes

- The loading is determined by file type.

from_string(*schema_string*, *schema_format='.xml'*, *schema_namespace=None*, *schema=None*, *name=None*)

Create a schema from the given string.

Parameters

- **schema_string** (*str*) – An XML or mediawiki file as a single long string
- **schema_format** (*str*) – The schema format of the source schema string. Allowed normal values: .mediawiki, .xml
- **schema_namespace** (*str, None*) – The name_prefix all tags in this schema will accept.
- **schema** (*HedSchema or None*) – A HED schema to merge this new file into It must be a with-standard schema with the same value.
- **name** (*str or None*) – User supplied identifier for this schema

Returns
The loaded schema.

Return type
(HedSchema)

Raises
HedFileError –

- If empty string or invalid extension is passed.
- Other fatal formatting issues with file

Notes

- The loading is determined by file type.

get_hed_xml_version(*xml_file_path*)

Get the version number from a HED XML file.

Parameters
xml_file_path (*str*) – The path to a HED XML file.

Returns
The version number of the HED XML file.

Return type
str

Raises
HedFileError –

- There is an issue loading the schema

load_schema(*hed_path*, *schema_namespace=None*, *schema=None*, *name=None*)

Load a schema from the given file or URL path.

Parameters

- **hed_path** (*str*) – A filepath or url to open a schema from. If loading a TSV file, this should be a single filename where: Template: basename.tsv, where files are named basename_Struct.tsv, basename_Tag.tsv, etc. Alternatively, you can point to a directory containing the .tsv files.
- **schema_namespace** (*str or None*) – The name_prefix all tags in this schema will accept.
- **schema** (*HedSchema or None*) – A HED schema to merge this new file into It must be a with-standard schema with the same value.
- **name** (*str or None*) – User supplied identifier for this schema

Returns
The loaded schema.

Return type
HedSchema

Raises
HedFileError –

- Empty path passed
- Unknown extension

- Any fatal issues when loading the schema.

load_schema_version(*xml_version=None*, *xml_folder=None*)

Return a HedSchema or HedSchemaGroup extracted from xml_version

Parameters

- **xml_version** (*str or list*) – List or str specifying which official HED schemas to use. A json str format is also supported, based on the output of HedSchema.get_formatted_version Basic format: *[schema_namespace:][library_name_]X.Y.Z.*
- **xml_folder** (*str*) – Path to a folder containing schema.

Returns
The schema or schema group extracted.

Return type
HedSchema or HedSchemaGroup

Raises
HedFileError –

- The xml_version is not valid.
- The specified version cannot be found or loaded
- Other fatal errors loading the schema (These are unlikely if you are not editing them locally)
- The prefix is invalid

parse_version_list(*xml_version_list*)

Takes a list of xml versions and returns a dictionary split by prefix

e.g. ["score", "testlib"] will return {"": "score, testlib"} e.g. ["score", "testlib", "ol:otherlib"] will return {"": "score, testlib", "ol:": "otherlib"}

Parameters
xml_version_list (*list*) – List of str specifying which HED schemas to use

Returns
The schema or schema group extracted.

Return type
HedSchema or HedSchemaGroup

3.3.9 hed_schema_section

Classes

HedSchemaSection(section_key[, case_sensitive])	Container with entries in one section of the schema.
HedSchemaTagSection(*args[, case_sensitive])	The schema section containing all tags.
HedSchemaUnitClassSection(section_key[, ...])	The schema section containing unit classes.
HedSchemaUnitSection(section_key[, ...])	The schema section containing units.

3.3.9.1 HedSchemaSection

class HedSchemaSection(*section_key*, *case_sensitive=True*)

Container with entries in one section of the schema.

Methods

HedSchemaSection.__init__(section_key[, ...])	Construct schema section.
HedSchemaSection.get(key)	Return the name associated with key.
HedSchemaSection.get_entries_with_attribute(...)	Return entries or names with given attribute.
HedSchemaSection.items()	Return the items.
HedSchemaSection.keys()	The names of the keys.
HedSchemaSection.values()	All names of the sections.

Attributes

HedSchemaSection.duplicate_names
HedSchemaSection.section_key

HedSchemaSection.**__init__**(*section_key*, *case_sensitive=True*)

Construct schema section.

Parameters

- **section_key** (*HedSectionKey*) – Name of the schema section.
- **case_sensitive** (*bool*) – If True, names are case-sensitive.

HedSchemaSection.**get**(*key*)

Return the name associated with key.

Parameters

key (*str*) – The name of the key.

HedSchemaSection.**get_entries_with_attribute**(*attribute_name*, *return_name_only=False*, *schema_namespace=''*)

Return entries or names with given attribute.

Parameters

- **attribute_name** (*str*) – The name of the attribute(generally a HedKey entry).
- **return_name_only** (*bool*) – If True, return the name as a string rather than the tag entry.
- **schema_namespace** (*str*) – Prepends given namespace to each name if returning names.

Returns

List of HedSchemaEntry or strings representing the names.

Return type

list

HedSchemaSection.**items**()

Return the items.

HedSchemaSection.**keys**()

The names of the keys.

HedSchemaSection.**values**()

All names of the sections.

HedSchemaSection.**duplicate_names**

HedSchemaSection.**section_key**

3.3.9.2 HedSchemaTagSection

class HedSchemaTagSection(**args*, *case_sensitive=False*, ***kwargs*)

The schema section containing all tags.

Methods

HedSchemaTagSection.__init__(*args[, ...])	Construct schema section.
HedSchemaTagSection.get(key)	Return the name associated with key.
HedSchemaTagSection.get_entries_with_attribute(...)	Return entries or names with given attribute.
HedSchemaTagSection.items()	Return the items.
HedSchemaTagSection.keys()	The names of the keys.
HedSchemaTagSection.values()	All names of the sections.

Attributes

HedSchemaTagSection.duplicate_names
HedSchemaTagSection.section_key

HedSchemaTagSection.**__init__**(**args*, *case_sensitive=False*, ***kwargs*)

Construct schema section.

Parameters

- **section_key** (*HedSectionKey*) – Name of the schema section.
- **case_sensitive** (*bool*) – If True, names are case-sensitive.

HedSchemaTagSection.**get**(*key*)

Return the name associated with key.

Parameters

key (*str*) – The name of the key.

HedSchemaTagSection.**get_entries_with_attribute**(*attribute_name*, *return_name_only=False*, *schema_namespace=''*)

Return entries or names with given attribute.

Parameters

- **attribute_name** (*str*) – The name of the attribute(generally a HedKey entry).
- **return_name_only** (*bool*) – If True, return the name as a string rather than the tag entry.
- **schema_namespace** (*str*) – Prepends given namespace to each name if returning names.

Returns
List of HedSchemaEntry or strings representing the names.

Return type
list

HedSchemaTagSection.**items**()

Return the items.

HedSchemaTagSection.**keys**()

The names of the keys.

HedSchemaTagSection.**values**()

All names of the sections.

HedSchemaTagSection.**duplicate_names**

HedSchemaTagSection.**section_key**

3.3.9.3 HedSchemaUnitClassSection

class HedSchemaUnitClassSection(*section_key*, *case_sensitive=True*)

The schema section containing unit classes.

Methods

HedSchemaUnitClassSection.__init__(section_key)	Construct schema section.
HedSchemaUnitClassSection.get(key)	Return the name associated with key.
HedSchemaUnitClassSection.get_entries_with_attribute(...)	Return entries or names with given attribute.
HedSchemaUnitClassSection.items()	Return the items.
HedSchemaUnitClassSection.keys()	The names of the keys.
HedSchemaUnitClassSection.values()	All names of the sections.

Attributes

HedSchemaUnitClassSection.duplicate_names

HedSchemaUnitClassSection.section_key

HedSchemaUnitClassSection.**__init__**(*section_key*, *case_sensitive=True*)

Construct schema section.

Parameters

- **section_key** (*HedSectionKey*) – Name of the schema section.
- **case_sensitive** (*bool*) – If True, names are case-sensitive.

HedSchemaUnitClassSection.**get**(*key*)

Return the name associated with key.

Parameters
key (*str*) – The name of the key.

HedSchemaUnitClassSection.**get_entries_with_attribute**(*attribute_name*, *return_name_only=False*, *schema_namespace=''*)

Return entries or names with given attribute.

Parameters

- **attribute_name** (*str*) – The name of the attribute(generally a HedKey entry).
- **return_name_only** (*bool*) – If True, return the name as a string rather than the tag entry.
- **schema_namespace** (*str*) – Prepends given namespace to each name if returning names.

Returns
List of HedSchemaEntry or strings representing the names.

Return type
list

HedSchemaUnitClassSection.**items**()

Return the items.

HedSchemaUnitClassSection.**keys**()

The names of the keys.

HedSchemaUnitClassSection.**values**()

All names of the sections.

HedSchemaUnitClassSection.**duplicate_names**

HedSchemaUnitClassSection.**section_key**

3.3.9.4 HedSchemaUnitSection

class HedSchemaUnitSection(*section_key*, *case_sensitive=True*)

The schema section containing units.

Methods

HedSchemaUnitSection.__init__(section_key[, ...])	Construct schema section.
HedSchemaUnitSection.get(key)	Return the name associated with key.
HedSchemaUnitSection.get_entries_with_attribute(...)	Return entries or names with given attribute.
HedSchemaUnitSection.items()	Return the items.
HedSchemaUnitSection.keys()	The names of the keys.
HedSchemaUnitSection.values()	All names of the sections.

Attributes

HedSchemaUnitSection.duplicate_names
HedSchemaUnitSection.section_key

HedSchemaUnitSection.**__init__**(*section_key*, *case_sensitive=True*)

Construct schema section.

Parameters

- **section_key** (*HedSectionKey*) – Name of the schema section.
- **case_sensitive** (*bool*) – If True, names are case-sensitive.

HedSchemaUnitSection.**get**(*key*)

Return the name associated with key.

Parameters
key (*str*) – The name of the key.

HedSchemaUnitSection.**get_entries_with_attribute**(*attribute_name*, *return_name_only=False*, *schema_namespace=''*)

Return entries or names with given attribute.

Parameters

- **attribute_name** (*str*) – The name of the attribute(generally a HedKey entry).
- **return_name_only** (*bool*) – If True, return the name as a string rather than the tag entry.
- **schema_namespace** (*str*) – Prepends given namespace to each name if returning names.

Returns
List of HedSchemaEntry or strings representing the names.

Return type
list

HedSchemaUnitSection.**items**()

Return the items.

HedSchemaUnitSection.**keys**()

The names of the keys.

HedSchemaUnitSection.**values**()

All names of the sections.

HedSchemaUnitSection.**duplicate_names**

HedSchemaUnitSection.**section_key**

3.3.10 schema_attribute_validator_hed_id

Classes

HedIDValidator(hed_schema)	Support class to validate hedIds in schemas

3.3.10.1 HedIDValidator

class HedIDValidator(*hed_schema*)

Support class to validate hedIds in schemas

Methods

HedIDValidator.__init__(hed_schema)	Support class to validate hedIds in schemas
HedIDValidator.verify_tag_id(hed_schema, ...)	Validates the hedID attribute values

Attributes

HedIDValidator.**__init__**(*hed_schema*)

Support class to validate hedIds in schemas

Parameters

- **hed_schema** (*HedSchemaBase*) – The schema we're validating.
- **number** (*It uses this to derive the version*) –

HedIDValidator.**verify_tag_id**(*hed_schema*, *tag_entry*, *attribute_name*)

Validates the hedID attribute values

This follows the template from schema_attribute_validators.py

Parameters

- **hed_schema** (*HedSchema*) – The schema to use for validation
- **tag_entry** (*HedSchemaEntry*) – The schema entry for this tag.

- **attribute_name** (*str*) – The name of this attribute

Returns
A list of issues from validating this attribute.

Return type
issues(list)

3.3.11 schema_attribute_validators

The built-in functions to validate known attributes.

Template for the functions:

- `attribute_checker_template(hed_schema, tag_entry, attribute_name): - hed_schema (HedSchema): The schema to use for validation. - tag_entry (HedSchemaEntry): The schema entry for this tag. - attribute_name (str): The name of this attribute.`

  returns
  A list of issues found validating this attribute

  rtype
  - `issues (list)`

Functions

allowed_characters_check(hed_schema, ...)	Check allowed character has a valid value
attribute_is_deprecated(hed_schema, ...)	Check if the attribute is deprecated.
conversion_factor(hed_schema, tag_entry, ...)	Check if the conversion_factor on is valid
in_library_check(hed_schema, tag_entry, ...)	Check if the library attribute is a valid schema name
is_numeric_value(hed_schema, tag_entry, ...)	Check if the attribute is a valid numeric(float) value
item_exists_check(hed_schema, tag_entry, ...)	Check if the list of possible items exists in the schema and are not deprecated.
tag_exists_base_schema_check(hed_schema, ...)	Check if the single tag is a partnered schema tag
tag_is_deprecated_check(hed_schema, ...)	Check if the element has a valid deprecatedFrom attribute, and that any children have it
tag_is_placeholder_check(hed_schema, ...)	Check if comma separated list has valid HedTags.
unit_exists(hed_schema, tag_entry, ...)	Check the given unit is valid, and not deprecated.

allowed_characters_check(*hed_schema*, *tag_entry*, *attribute_name*)

Check allowed character has a valid value

Parameters
- **hed_schema** (*HedSchema*) – The schema to use for validation
- **tag_entry** (*HedSchemaEntry*) – The schema entry for this tag.
- **attribute_name** (*str*) – The name of this attribute

Returns
A list of issues from validating this attribute.

Return type
issues(list)

attribute_is_deprecated(*hed_schema*, *tag_entry*, *attribute_name*)

Check if the attribute is deprecated. does not check value.

Parameters

- **hed_schema** (*HedSchema*) – The schema to use for validation
- **tag_entry** (*HedSchemaEntry*) – The schema entry for this tag.
- **attribute_name** (*str*) – The name of this attribute

Returns
A list of issues from validating this attribute.

Return type
issues(list)

conversion_factor(*hed_schema*, *tag_entry*, *attribute_name*)

Check if the conversion_factor on is valid

Parameters

- **hed_schema** (*HedSchema*) – The schema to use for validation
- **tag_entry** (*HedSchemaEntry*) – The schema entry for this tag.
- **attribute_name** (*str*) – The name of this attribute

Returns
A list of issues from validating this attribute.

Return type
issues(list)

in_library_check(*hed_schema*, *tag_entry*, *attribute_name*)

Check if the library attribute is a valid schema name

Parameters

- **hed_schema** (*HedSchema*) – The schema to use for validation
- **tag_entry** (*HedSchemaEntry*) – The schema entry for this tag.
- **attribute_name** (*str*) – The name of this attribute

Returns
A list of issues from validating this attribute.

Return type
issues(list)

is_numeric_value(*hed_schema*, *tag_entry*, *attribute_name*)

Check if the attribute is a valid numeric(float) value

Parameters

- **hed_schema** (*HedSchema*) – The schema to use for validation
- **tag_entry** (*HedSchemaEntry*) – The schema entry for this tag.
- **attribute_name** (*str*) – The name of this attribute

Returns
A list of issues from validating this attribute.

Return type
issues(list)

item_exists_check(*hed_schema*, *tag_entry*, *attribute_name*, *section_key*)

Check if the list of possible items exists in the schema and are not deprecated.

Parameters

- **hed_schema** (*HedSchema*) – The schema to use for validation
- **tag_entry** (*HedSchemaEntry*) – The schema entry for this tag.
- **attribute_name** (*str*) – The name of this attribute
- **section_key** (*HedSectionKey*) – The section this item should be in. This is generally passed via functools.partial

Returns

A list of issues from validating this attribute.

Return type

issues(list)

tag_exists_base_schema_check(*hed_schema*, *tag_entry*, *attribute_name*)

Check if the single tag is a partnered schema tag

Parameters

- **hed_schema** (*HedSchema*) – The schema to use for validation
- **tag_entry** (*HedSchemaEntry*) – The schema entry for this tag.
- **attribute_name** (*str*) – The name of this attribute

Returns

A list of issues from validating this attribute.

Return type

issues(list)

tag_is_deprecated_check(*hed_schema*, *tag_entry*, *attribute_name*)

Check if the element has a valid deprecatedFrom attribute, and that any children have it

Parameters

- **hed_schema** (*HedSchema*) – The schema to use for validation
- **tag_entry** (*HedSchemaEntry*) – The schema entry for this tag.
- **attribute_name** (*str*) – The name of this attribute

Returns

A list of issues from validating this attribute.

Return type

issues(list)

tag_is_placeholder_check(*hed_schema*, *tag_entry*, *attribute_name*)

Check if comma separated list has valid HedTags.

Parameters

- **hed_schema** (*HedSchema*) – The schema to use for validation
- **tag_entry** (*HedSchemaEntry*) – The schema entry for this tag.
- **attribute_name** (*str*) – The name of this attribute

Returns

A list of issues from validating this attribute.

Return type
issues(list)

unit_exists(*hed_schema*, *tag_entry*, *attribute_name*)

Check the given unit is valid, and not deprecated.

Parameters

- **hed_schema** (*HedSchema*) – The schema to use for validation
- **tag_entry** (*HedSchemaEntry*) – The schema entry for this tag.
- **attribute_name** (*str*) – The name of this attribute

Returns
A list of issues from validating this attribute.

Return type
issues(list)

3.3.12 schema_compare

Functions supporting comparison of schemas.

Functions

compare_differences(schema1, schema2[, ...])	Compare the tags in two schemas, this finds any differences
compare_schemas(schema1, schema2[, ...])	Compare two schemas section by section.
find_matching_tags(schema1, schema2[, ...])	Compare the tags in two library schemas.
gather_schema_changes(schema1, schema2[, ...])	Compare two schemas section by section, generating a changelog
pretty_print_change_dict(change_dict[, ...])	Formats the change_dict into a string.

compare_differences(*schema1*, *schema2*, *attribute_filter=None*, *title=''*)

Compare the tags in two schemas, this finds any differences

Parameters

- **schema1** (*HedSchema*) – The first schema to be compared.
- **schema2** (*HedSchema*) – The second schema to be compared.
- **attribute_filter** (*str, optional*) – The attribute to filter entries by. Entries without this attribute are skipped. The most common use would be HedKey.InLibrary If it evaluates to False, no filtering is performed.
- **title** (*str*) – Optional header to add, a default on will be added otherwise.

Returns
the changes listed out by section

Return type
changelog(str)

compare_schemas(*schema1*, *schema2*, *attribute_filter='inLibrary'*, *sections=(<HedSectionKey.Tags: 'tags'>,)*)

Compare two schemas section by section.

The function records matching entries, entries present in one schema but not in the other, and unequal entries.

Parameters

- **schema1** (*HedSchema*) – The first schema to be compared.
- **schema2** (*HedSchema*) – The second schema to be compared.
- **attribute_filter** (*str, optional*) – The attribute to filter entries by. Entries without this attribute are skipped. The most common use would be HedKey.InLibrary If it evaluates to False, no filtering is performed.
- **sections** (*list or None*) – the list of sections to compare. By default, just the tags section. If None, checks all sections including header, prologue, and epilogue.

Returns

A tuple containing four dictionaries: - matches(dict): Entries present in both schemas and are equal. - not_in_schema1(dict): Entries present in schema2 but not in schema1. - not_in_schema2(dict): Entries present in schema1 but not in schema2. - unequal_entries(dict): Entries present in both schemas but are not equal.

Return type

tuple

find_matching_tags(*schema1*, *schema2*, *sections=(<HedSectionKey.Tags: 'tags'>,)*, *return_string=True*)

Compare the tags in two library schemas. This finds tags with the same term.

Parameters

- **schema1** (*HedSchema*) – The first schema to be compared.
- **schema2** (*HedSchema*) – The second schema to be compared.
- **sections** (*list*) – the list of sections to compare. By default, just the tags section. If None, checks all sections including header, prologue, and epilogue.
- **return_string** (*bool*) – If False, returns the raw python dictionary(for tools etc. possible use)

Returns

Returns a formatted string or python dict

Return type

str or dict

gather_schema_changes(*schema1*, *schema2*, *attribute_filter=None*)

Compare two schemas section by section, generating a changelog

Parameters

- **schema1** (*HedSchema*) – The first schema to be compared.
- **schema2** (*HedSchema*) – The second schema to be compared.
- **attribute_filter** (*str, optional*) – The attribute to filter entries by. Entries without this attribute are skipped. The most common use would be HedKey.InLibrary If it evaluates to False, no filtering is performed.

Returns

A dict organized by section with the changes

Return type
changelog(dict)

pretty_print_change_dict(*change_dict*, *title='Schema changes'*, *use_markdown=True*)

Formats the change_dict into a string.

Parameters

- **change_dict** (*dict*) – The result from calling gather_schema_changes
- **title** (*str*) – Optional header to add, a default on will be added otherwise.
- **use_markdown** (*bool*) – If True, adds Markdown formatting characters to output.

Returns
the changes listed out by section

Return type
changelog(str)

3.3.13 schema_compliance

Utilities for HED schema checking.

Functions

check_compliance(hed_schema[, ...])	Check for hed3 compliance of a schema object.

check_compliance(*hed_schema*, *check_for_warnings=True*, *name=None*, *error_handler=None*)

Check for hed3 compliance of a schema object.

Parameters

- **hed_schema** (*HedSchema*) – HedSchema object to check for hed3 compliance.
- **check_for_warnings** (*bool*) – If True, check for formatting issues like invalid characters, capitalization, etc.
- **name** (*str*) – If present, will use as filename for context.
- **error_handler** (*ErrorHandler or None*) – Used to report errors. Uses a default one if none passed in.

Returns
A list of all warnings and errors found in the file. Each issue is a dictionary.

Return type
list

Raises
ValueError –

- Trying to validate a HedSchemaGroup directly

Classes

SchemaValidator(hed_schema, error_handler)	Validator class to wrap some code.

3.3.13.1 SchemaValidator

class SchemaValidator(*hed_schema*, *error_handler*)

Validator class to wrap some code. In general, just call check_compliance.

Methods

SchemaValidator.__init__(hed_schema, ...)	
SchemaValidator.check_attributes()	Returns issues from validating known attributes in all sections
SchemaValidator.check_duplicate_names()	Return issues for any duplicate names in all sections.
SchemaValidator.check_if_prerelease_version()	
SchemaValidator.check_invalid_chars()	Returns issues for bad chars in terms or descriptions.
SchemaValidator.check_prologue_epilogue()	

Attributes

SchemaValidator.attribute_validators
SchemaValidator.attribute_validators_old

SchemaValidator.**__init__**(*hed_schema*, *error_handler*)

SchemaValidator.**check_attributes**()

Returns issues from validating known attributes in all sections

SchemaValidator.**check_duplicate_names**()

Return issues for any duplicate names in all sections.

SchemaValidator.**check_if_prerelease_version**()

SchemaValidator.**check_invalid_chars**()

Returns issues for bad chars in terms or descriptions.

SchemaValidator.**check_prologue_epilogue**()

SchemaValidator.**attribute_validators** = {'allowedCharacter': [<function allowed_characters_check>], 'conversionFactor': [<function conversion_factor>], 'defaultUnits': [], 'deprecatedFrom': [<function tag_is_deprecated_check>], 'inLibrary': [<function in_library_check>], 'relatedTag': [], 'suggestedTag': [], 'takesValue': [<function tag_is_placeholder_check>], 'unitClass': [<function tag_is_placeholder_check>], 'valueClass': [<function tag_is_placeholder_check>]}

SchemaValidator.attribute_validators_old = {'allowedCharacter': [<function allowed_characters_check>], 'conversionFactor': [<function conversion_factor>], 'defaultUnits': [<function unit_exists>], 'deprecatedFrom': [<function tag_is_deprecated_check>], 'inLibrary': [<function in_library_check>], 'relatedTag': [functools.partial(<function item_exists_check>, section_key=<HedSectionKey.Tags: 'tags'>)], 'suggestedTag': [functools.partial(<function item_exists_check>, section_key=<HedSectionKey.Tags: 'tags'>)], 'takesValue': [<function tag_is_placeholder_check>], 'unitClass': [<function tag_is_placeholder_check>, functools.partial(<function item_exists_check>, section_key=<HedSectionKey.UnitClasses: 'unitClasses'>)], 'valueClass': [<function tag_is_placeholder_check>, functools.partial(<function item_exists_check>, section_key=<HedSectionKey.ValueClasses: 'valueClasses'>)]}

3.3.14 schema_header_util

Functions

validate_attributes(attrib_dict, name)	Validate attributes in the dictionary.
validate_library_name(library_name)	Check the validity of the library name.
validate_present_attributes(attrib_dict, name)	Validate combinations of attributes
validate_version_string(version_string)	Check validity of the version.

validate_attributes(*attrib_dict*, *name*)

Validate attributes in the dictionary.

Parameters

- **attrib_dict** (*dict*) – Dictionary of attributes to be evaluated.
- **name** (*str*) – name to use in reporting errors.

Returns

List of issues. Each issue is a dictionary.

Return type

list

Raises

HedFileError –

- Invalid library name
- Version not present
- Invalid combinations of attributes in header

validate_library_name(*library_name*)

Check the validity of the library name.

Parameters

library_name (*str*) – Name of the library.

Returns

If not False, string indicates the issue.

Return type

bool or str

validate_present_attributes(*attrib_dict*, *name*)

Validate combinations of attributes

Parameters

- **attrib_dict** (*dict*) – Dictionary of attributes to be evaluated.
- **name** (*str*) – File name to use in reporting errors.

Returns

List of issues. Each issue is a dictionary.

Return type

list

Raises

HedFileError –

- withStandard is found in th header, but a library attribute is not specified

validate_version_string(*version_string*)

Check validity of the version.

Parameters

version_string (*str*) – A version string.

Returns

If not False, string indicates the issue.

Return type

bool or str

3.3.15 schema_io

Modules

hed.schema.schema_io.base2schema	
hed.schema.schema_io.df2schema	This module is used to create a HedSchema object from a set of .tsv files.
hed.schema.schema_io.df_constants	
hed.schema.schema_io.df_util	
hed.schema.schema_io.ontology_util	Utility functions for saving as an ontology or dataframe.
hed.schema.schema_io.schema2base	Baseclass for mediawiki/xml writers
hed.schema.schema_io.schema2df	Allows output of HedSchema objects as .tsv format
hed.schema.schema_io.schema2wiki	Allows output of HedSchema objects as .mediawiki format
hed.schema.schema_io.schema2xml	Allows output of HedSchema objects as .xml format
hed.schema.schema_io.schema_util	Utilities for writing content to files and for other file manipulation.
hed.schema.schema_io.text_util	Functions for parsing text from dataframes/text formats
hed.schema.schema_io.wiki2schema	This module is used to create a HedSchema object from a .mediawiki file.
hed.schema.schema_io.wiki_constants	
hed.schema.schema_io.xml2schema	This module is used to create a HedSchema object from an XML file or tree.
hed.schema.schema_io.xml_constants	Constants used for the

3.3.15.1 base2schema

Classes

SchemaLoader(filename[, schema_as_string, ...])	Baseclass for schema loading, to handle basic errors and partnered schemas

3.3.15.1.1 SchemaLoader

class SchemaLoader(*filename*, *schema_as_string=None*, *schema=None*, *file_format=None*, *name=''*)

Baseclass for schema loading, to handle basic errors and partnered schemas

Expected usage is SchemaLoaderXML.load(filename)

SchemaLoaderXML(filename) will load just the header_attributes

Methods

SchemaLoader.__init__(filename[, ...])	Loads the given schema from one of the two parameters.
SchemaLoader.find_rooted_entry(tag_entry, ...)	This semi-validates rooted tags, raising an exception on major errors
SchemaLoader.fix_extra(key)	
SchemaLoader.fix_extras()	Fixes the extras after loading the schema, to ensure they are in the correct format.
SchemaLoader.load([filename, ...])	Loads and returns the schema, including partnered schema if applicable.

Attributes

SchemaLoader.schema	The partially loaded schema if you are after just header attributes.

SchemaLoader.**__init__**(*filename*, *schema_as_string=None*, *schema=None*, *file_format=None*, *name=''*)

Loads the given schema from one of the two parameters.

Parameters

- **filename** (*str or None*) – A valid filepath or None
- **schema_as_string** (*str or None*) – A full schema as text or None
- **schema** (*HedSchema or None*) – A HED schema to merge this new file into It must be a with-standard schema with the same value.
- **file_format** (*str or None*) – The format of this file if needed(only for owl currently)
- **name** (*str or None*) – Optional user supplied identifier, by default uses filename

static SchemaLoader.**find_rooted_entry**(*tag_entry*, *schema*, *loading_merged*)

This semi-validates rooted tags, raising an exception on major errors

Parameters

- **tag_entry** (*HedTagEntry*) – the possibly rooted tag
- **schema** (*HedSchema*) – The schema being loaded
- **loading_merged** (*bool*) – If this schema was already merged before loading

Returns

The base tag entry from the standard schema
Returns None if this tag isn't rooted

Return type
rooted_tag(HedTagEntry or None)

Raises
HedFileError –

- A rooted attribute is found in a non-paired schema
- A rooted attribute is not a string
- A rooted attribute was found on a non-root node in an unmerged schema.

- A rooted attribute is found on a root node in a merged schema.
- A rooted attribute indicates a tag that doesn't exist in the base schema.

SchemaLoader.**fix_extra**(*key*)

SchemaLoader.**fix_extras**()

Fixes the extras after loading the schema, to ensure they are in the correct format.

classmethod SchemaLoader.**load**(*filename=None*, *schema_as_string=None*, *schema=None*, *file_format=None*, *name=''*)

Loads and returns the schema, including partnered schema if applicable.

Parameters

- **filename** (*str or None*) – A valid filepath or None
- **schema_as_string** (*str or None*) – A full schema as text or None
- **schema** (*HedSchema or None*) – A HED schema to merge this new file into It must be a with-standard schema with the same value.
- **file_format** (*str or None*) – If this is an owl file being loaded, this is the format. Allowed values include: turtle, json-ld, and owl(xml)
- **name** (*str or None*) – Optional user supplied identifier, by default uses filename

Returns
The new schema

Return type
schema(HedSchema)

SchemaLoader.**schema**

The partially loaded schema if you are after just header attributes.

3.3.15.2 df2schema

This module is used to create a HedSchema object from a set of .tsv files.

Functions

load_dataframes_from_strings(schema_data)	Load the given strings/dataframes as dataframes.

load_dataframes_from_strings(*schema_data*)

Load the given strings/dataframes as dataframes.

Parameters
schema_data (*dict*) – The dict of files(strings or pd.DataFrames) key being constants like TAG_KEY

Returns
A dict with the same keys as schema_data, but values are dataframes if not before

Return type
schema_data(dict)

Classes

SchemaLoaderDF(filenames, ...[, name])	Load dataframe schemas from filenames

3.3.15.2.1 SchemaLoaderDF

class SchemaLoaderDF(*filenames*, *schema_as_strings_or_df*, *name=''*)

Load dataframe schemas from filenames

Expected usage is SchemaLoaderDF.load(filenames)

Note: due to supporting multiple files, this one differs from the other schema loaders

Methods

SchemaLoaderDF.__init__(filenames, ...[, name])	Loads the given schema from one of the two parameters.
SchemaLoaderDF.find_rooted_entry(tag_entry, ...)	This semi-validates rooted tags, raising an exception on major errors
SchemaLoaderDF.fix_extra(key)	
SchemaLoaderDF.fix_extras()	Fixes the extras after loading the schema, to ensure they are in the correct format.
SchemaLoaderDF.load([filename, ...])	Loads and returns the schema, including partnered schema if applicable.
SchemaLoaderDF.load_spreadsheet([filenames, ...])	Loads and returns the schema, including partnered schema if applicable.

Attributes

SchemaLoaderDF.schema	The partially loaded schema if you are after just header attributes.

SchemaLoaderDF.**__init__**(*filenames*, *schema_as_strings_or_df*, *name=''*)

Loads the given schema from one of the two parameters.

Parameters

- **filename** (*str or None*) – A valid filepath or None
- **schema_as_string** (*str or None*) – A full schema as text or None
- **schema** (*HedSchema or None*) – A HED schema to merge this new file into It must be a with-standard schema with the same value.
- **file_format** (*str or None*) – The format of this file if needed(only for owl currently)
- **name** (*str or None*) – Optional user supplied identifier, by default uses filename

static SchemaLoaderDF.**find_rooted_entry**(*tag_entry*, *schema*, *loading_merged*)

This semi-validates rooted tags, raising an exception on major errors

Parameters

- **tag_entry** (*HedTagEntry*) – the possibly rooted tag
- **schema** (*HedSchema*) – The schema being loaded
- **loading_merged** (*bool*) – If this schema was already merged before loading

Returns

The base tag entry from the standard schema
Returns None if this tag isn't rooted

Return type
rooted_tag(HedTagEntry or None)

Raises
HedFileError –

- A rooted attribute is found in a non-paired schema
- A rooted attribute is not a string
- A rooted attribute was found on a non-root node in an unmerged schema.
- A rooted attribute is found on a root node in a merged schema.
- A rooted attribute indicates a tag that doesn't exist in the base schema.

SchemaLoaderDF.**fix_extra**(*key*)

SchemaLoaderDF.**fix_extras**()

Fixes the extras after loading the schema, to ensure they are in the correct format.

classmethod SchemaLoaderDF.**load**(*filename=None*, *schema_as_string=None*, *schema=None*, *file_format=None*, *name=''*)

Loads and returns the schema, including partnered schema if applicable.

Parameters

- **filename** (*str or None*) – A valid filepath or None
- **schema_as_string** (*str or None*) – A full schema as text or None
- **schema** (*HedSchema or None*) – A HED schema to merge this new file into It must be a with-standard schema with the same value.
- **file_format** (*str or None*) – If this is an owl file being loaded, this is the format. Allowed values include: turtle, json-ld, and owl(xml)
- **name** (*str or None*) – Optional user supplied identifier, by default uses filename

Returns
The new schema

Return type
schema(HedSchema)

classmethod SchemaLoaderDF.**load_spreadsheet**(*filenames=None*, *schema_as_strings_or_df=None*, *name=''*)

Loads and returns the schema, including partnered schema if applicable.

Parameters

- **filenames** (*str or None or dict of str*) – A valid set of schema spreadsheet filenames If a single filename string, assumes the standard filename suffixes.

- **schema_as_strings_or_df** (*None or dict of str*) – A valid set of schema spreadsheet files(tsv as strings)
- **name** (*str*) – what to identify this schema as

Returns
The new schema

Return type
schema(HedSchema)

SchemaLoaderDF.**schema**

The partially loaded schema if you are after just header attributes.

3.3.15.3 df_constants

3.3.15.4 df_util

Functions

calculate_attribute_type(attribute_entry)	Returns the type of this attribute(annotation, object, data)
convert_filenames_to_dict(filenames)	Infers filename meaning based on suffix, e.g.
create_empty_dataframes()	Returns the default empty dataframes
get_attributes_from_row(row)	Get the tag attributes from a line.
get_library_name_and_id(schema)	Get the library("Standard" for the standard schema) and first id for a schema range
load_dataframes(filenames)	Load the dataframes from the source folder or series of files.
merge_dataframe_dicts(df_dict1, df_dict2[, ...])	Create a new dictionary of DataFrames where dict2 is merged into dict1.
merge_dataframes(df1, df2, key)	Create a new dataframe where df2 is merged into df1 and duplicates are eliminated.
remove_prefix(text, prefix)	
save_dataframes(base_filename, dataframe_dict)	Writes out the dataframes using the provided suffixes.

calculate_attribute_type(*attribute_entry*)

Returns the type of this attribute(annotation, object, data)

Returns
"annotation", "object", or "data".

Return type
attribute_type(str)

convert_filenames_to_dict(*filenames*)

Infers filename meaning based on suffix, e.g. _Tag for the tags sheet

Parameters
filenames (*str or None or list or dict*) – The list to convert to a dict If a string with a .tsv suffix: Save to that location, adding the suffix to each .tsv file If a string with no .tsv suffix: Save to that folder, with the contents being the separate .tsv files.

Returns
str): The required suffix to filename mapping

Return type
filename_dict(str

`create_empty_dataframes()`

Returns the default empty dataframes

`get_attributes_from_row(row)`

Get the tag attributes from a line.

Parameters
row (*pd.Series*) – A tag line.

Returns
Dictionary of attributes.

Return type
dict

`get_library_name_and_id(schema)`

Get the library("Standard" for the standard schema) and first id for a schema range

Parameters
schema (*HedSchema*) – The schema to check

Returns
The capitalized library name first_id(int): the first id for a given library

Return type
library_name(str)

`load_dataframes(filenames)`

Load the dataframes from the source folder or series of files.

Parameters
filenames (*str or None or list or dict*) – The input filenames If a string with a .tsv suffix: Save to that location, adding the suffix to each .tsv file If a string with no .tsv suffix: Save to that folder, with the contents being the separate .tsv files.

Returns
dataframes): The suffix:dataframe dict

Return type
dataframes_dict(str

`merge_dataframe_dicts(df_dict1, df_dict2, key_column='rdfs.label')`

Create a new dictionary of DataFrames where dict2 is merged into dict1.

Does not validate contents or suffixes.

Parameters

- **str** (*df_dict2(dict of*) – df.DataFrame): dataframes to use as destination merge.
- **str** – df.DataFrame): dataframes to use as a merge element.
- **key_column** (*str*) – name of the column that is treated as the key when dataframes are merged

`merge_dataframes(df1, df2, key)`

Create a new dataframe where df2 is merged into df1 and duplicates are eliminated.

Parameters

- **df1** (*df.DataFrame*) – dataframe to use as destination merge.

- **df2** (*df.DataFrame*) – dataframe to use as a merge element.
- **key** (*str*) – name of the column that is treated as the key when dataframes are merged

Returns
The merged dataframe.

Return type
df.DataFrame

remove_prefix(*text*, *prefix*)

save_dataframes(*base_filename*, *dataframe_dict*)

Writes out the dataframes using the provided suffixes.

Does not validate contents or suffixes.

If base_filename has a .tsv suffix, save directly to the indicated location. If base_filename is a directory(does NOT have a .tsv suffix), save the contents into a directory named that. The subfiles are named the same. e.g. HED8.3.0/HED8.3.0_Tag.tsv

Parameters

- **base_filename** (*str*) – The base filename to use. Output is {base_filename}_{suffix}.tsv See DF_SUFFIXES for all expected names.
- **str** (*dataframe_dict(dict of*) – df.DataFrame): The list of files to save out. No validation is done.

3.3.15.5 ontology_util

Utility functions for saving as an ontology or dataframe.

Functions

assign_hed_ids_section(df, unused_tag_ids)	Adds missing HedIds to dataframe.
convert_df_to_omn(dataframes)	Convert the dataframe format schema to omn format.
get_all_ids(df)	Returns a set of all unique hedIds in the dataframe
get_prefixes(dataframes)	
merge_dfs(dest_df, source_df)	Merges extra columns from source_df into dest_df, adding the extra columns from the ontology to the schema df.
update_dataframes_from_schema(dataframes, schema)	Write out schema as a dataframe, then merge in extra columns from dataframes.

assign_hed_ids_section(*df*, *unused_tag_ids*)

Adds missing HedIds to dataframe.

Parameters

- **df** (*pd.DataFrame*) – The dataframe to add id's to.
- **unused_tag_ids** (*set of int*) – The possible HED id's to assign from

convert_df_to_omn(*dataframes*)

Convert the dataframe format schema to omn format.

Parameters
dataframes (*dict*) – A set of dataframes representing a schema, potentially including extra columns

Returns
omn_file(str): A combined string representing (most of) a schema omn file. omn_data(dict): a dict of DF_SUFFIXES:str, representing each .tsv file in omn format.

Return type
tuple

get_all_ids(*df*)

Returns a set of all unique hedIds in the dataframe

Parameters
df (*pd.DataFrame*) – The dataframe

Returns
None if this has no HED column, otherwise all unique numbers as a set.

Return type
numbers(Set or None)

get_prefixes(*dataframes*)

merge_dfs(*dest_df*, *source_df*)

Merges extra columns from source_df into dest_df, adding the extra columns from the ontology to the schema df.

Parameters

- **dest_df** – The dataframe to add extra columns to
- **source_df** – The dataframe to get extra columns from

update_dataframes_from_schema(*dataframes*, *schema*, *schema_name=''*, *get_as_ids=False*, *assign_missing_ids=False*)

Write out schema as a dataframe, then merge in extra columns from dataframes.

Parameters

- **dataframes** (*dict*) – A full set of schema spreadsheet formatted dataframes
- **schema** (*HedSchema*) – The schema to write into the dataframes:
- **schema_name** (*str*) – The name to use to find the schema id range.
- **get_as_ids** (*bool*) – If True, replace all known references with HedIds
- **assign_missing_ids** (*bool*) – If True, replacing any blank(new) HedIds with valid ones

Returns

pd.DataFrames): The updated dataframes
These dataframes can potentially have extra columns

Return type
dataframes(dict of str

3.3.15.6 schema2base

Baseclass for mediawiki/xml writers

Classes

Schema2Base()

3.3.15.6.1 Schema2Base

class Schema2Base

Methods

Schema2Base.__init__()	
Schema2Base.process_schema(hed_schema[, ...])	Takes a HedSchema object and returns it in the inherited form(mediawiki, xml, etc)

Attributes

Schema2Base.**__init__**()

Schema2Base.**process_schema**(*hed_schema*, *save_merged=False*)

Takes a HedSchema object and returns it in the inherited form(mediawiki, xml, etc)

Parameters

- **hed_schema** (*HedSchema*) – The schema to be processed.
- **save_merged** (*bool*) – If True, save as merged schema if has "withStandard".

Returns

Varies based on inherited class

Return type

Any

3.3.15.7 schema2df

Allows output of HedSchema objects as .tsv format

Classes

Schema2DF([get_as_ids])

3.3.15.7.1 Schema2DF

class Schema2DF(*get_as_ids=False*)

Methods

Schema2DF.__init__([get_as_ids])	Constructor for schema to dataframe converter
Schema2DF.process_schema(hed_schema[, ...])	Takes a HedSchema object and returns it in the inherited form(mediawiki, xml, etc)

Attributes

Schema2DF.**__init__**(*get_as_ids=False*)

Constructor for schema to dataframe converter

Parameters

get_as_ids (*bool*) – If true, return the hedId rather than name in most places This is mostly relevant for creating an ontology.

Schema2DF.**process_schema**(*hed_schema*, *save_merged=False*)

Takes a HedSchema object and returns it in the inherited form(mediawiki, xml, etc)

Parameters

- **hed_schema** (*HedSchema*) – The schema to be processed.
- **save_merged** (*bool*) – If True, save as merged schema if has "withStandard".

Returns

Varies based on inherited class

Return type

Any

3.3.15.8 schema2wiki

Allows output of HedSchema objects as .mediawiki format

Classes

Schema2Wiki()

3.3.15.8.1 Schema2Wiki

class Schema2Wiki

Methods

Schema2Wiki.__init__()	
Schema2Wiki.process_schema(hed_schema[, ...])	Takes a HedSchema object and returns it in the inherited form(mediawiki, xml, etc)

Attributes

Schema2Wiki.**__init__**()

Schema2Wiki.**process_schema**(*hed_schema*, *save_merged=False*)

Takes a HedSchema object and returns it in the inherited form(mediawiki, xml, etc)

Parameters

- **hed_schema** (*HedSchema*) – The schema to be processed.
- **save_merged** (*bool*) – If True, save as merged schema if has "withStandard".

Returns

Varies based on inherited class

Return type

Any

3.3.15.9 schema2xml

Allows output of HedSchema objects as .xml format

Classes

Schema2XML()

3.3.15.9.1 Schema2XML

class Schema2XML

Methods

Schema2XML.__init__()	
Schema2XML.process_schema(hed_schema[, ...])	Takes a HedSchema object and returns it in the inherited form(mediawiki, xml, etc)

Attributes

Schema2XML.**__init__**()

Schema2XML.**process_schema**(*hed_schema*, *save_merged=False*)

Takes a HedSchema object and returns it in the inherited form(mediawiki, xml, etc)

Parameters

- **hed_schema** (*HedSchema*) – The schema to be processed.
- **save_merged** (*bool*) – If True, save as merged schema if has "withStandard".

Returns

Varies based on inherited class

Return type

Any

3.3.15.10 schema_util

Utilities for writing content to files and for other file manipulation.

Functions

format_error(row_number, row[, ...])	

get_api_key()	Tries to get the GitHub access token from the environment. Defaults to above value if not found.
make_url_request(resource_url[, ...])	Make a request and adds the above GitHub access credentials.
schema_version_greater_equal(hed_schema, ...)	Check if the given schema standard version is above target version
url_to_file(resource_url)	Write data from a URL resource into a file.
url_to_string(resource_url)	Get the data from the specified url as a string.
xml_element_2_str(elem)	Convert an XML element to an XML string.

format_error(*row_number*, *row*, *warning_message='Schema term is empty or the line is malformed'*, *error_code='GENERIC_ERROR'*)

get_api_key()

Tries to get the GitHub access token from the environment. Defaults to above value if not found.

Returns
A GitHub access key or an empty string.

make_url_request(*resource_url*, *try_authenticate=True*)

Make a request and adds the above GitHub access credentials.

Parameters

- **resource_url** (*str*) – The url to retrieve.
- **try_authenticate** (*bool*) – If True add the above credentials.

Returns
url_request

schema_version_greater_equal(*hed_schema*, *target_version*)

Check if the given schema standard version is above target version

Parameters

- **hed_schema** (*HedSchema or HedSchemaGroup*) – If a schema group, checks if any version is above.
- **target_version** (*str*) – The semantic version to check against

Returns

True if the version is above target_version
False if it is not, or it is ambiguous.

Return type
bool

url_to_file(*resource_url*)

Write data from a URL resource into a file. Data is decoded as unicode.

Parameters
resource_url (*str*) – The URL to the resource.

Returns
The local temporary filename for the downloaded file,

Return type
str

url_to_string(*resource_url*)

Get the data from the specified url as a string.

Parameters
resource_url (*str*) – The URL to the resource.

Returns
The data at the target url.

Return type
str

xml_element_2_str(*elem*)

Convert an XML element to an XML string.

Parameters
elem (*Element*) – An XML element.

Returns
An XML string representing the XML element.

Return type
str

3.3.15.11 text_util

Functions for parsing text from dataframes/text formats

Functions

parse_attribute_string(attr_string)	Parse attributes for a single element into a dict.

parse_attribute_string(*attr_string*)

Parse attributes for a single element into a dict.

Parameters
attr_string (*str*) – Formatted attributes (a=b, c=d, etc.)

Returns
The located attributes. Can be empty.

Return type
attributes(dict)

Raises
ValueError –

- Very malformed input

3.3.15.12 wiki2schema

This module is used to create a HedSchema object from a .mediawiki file.

Classes

SchemaLoaderWiki(filename[, ...])	Load MediaWiki schemas from filenames or strings.

3.3.15.12.1 SchemaLoaderWiki

class SchemaLoaderWiki(*filename*, *schema_as_string=None*, *schema=None*, *file_format=None*, *name=''*)

Load MediaWiki schemas from filenames or strings.

Expected usage is SchemaLoaderWiki.load(filename)

SchemaLoaderWiki(filename) will load just the header_attributes

Methods

SchemaLoaderWiki.__init__(filename[, ...])	Loads the given schema from one of the two parameters.
SchemaLoaderWiki.find_rooted_entry(...)	This semi-validates rooted tags, raising an exception on major errors
SchemaLoaderWiki.fix_extra(key)	
SchemaLoaderWiki.fix_extras()	Fixes the extras after loading the schema, to ensure they are in the correct format.
SchemaLoaderWiki.load([filename, ...])	Loads and returns the schema, including partnered schema if applicable.
SchemaLoaderWiki.parse_star_string(s)	

Attributes

SchemaLoaderWiki.schema	The partially loaded schema if you are after just header attributes.

SchemaLoaderWiki.**__init__**(*filename*, *schema_as_string=None*, *schema=None*, *file_format=None*, *name=''*)

Loads the given schema from one of the two parameters.

Parameters

- **filename** (*str or None*) – A valid filepath or None
- **schema_as_string** (*str or None*) – A full schema as text or None
- **schema** (*HedSchema or None*) – A HED schema to merge this new file into It must be a with-standard schema with the same value.

- **file_format** (*str or None*) – The format of this file if needed(only for owl currently)
- **name** (*str or None*) – Optional user supplied identifier, by default uses filename

static SchemaLoaderWiki.**find_rooted_entry**(*tag_entry*, *schema*, *loading_merged*)

This semi-validates rooted tags, raising an exception on major errors

Parameters

- **tag_entry** (*HedTagEntry*) – the possibly rooted tag
- **schema** (*HedSchema*) – The schema being loaded
- **loading_merged** (*bool*) – If this schema was already merged before loading

Returns

The base tag entry from the standard schema
Returns None if this tag isn't rooted

Return type
rooted_tag(HedTagEntry or None)

Raises
HedFileError –

- A rooted attribute is found in a non-paired schema
- A rooted attribute is not a string
- A rooted attribute was found on a non-root node in an unmerged schema.
- A rooted attribute is found on a root node in a merged schema.
- A rooted attribute indicates a tag that doesn't exist in the base schema.

SchemaLoaderWiki.**fix_extra**(*key*)

SchemaLoaderWiki.**fix_extras**()

Fixes the extras after loading the schema, to ensure they are in the correct format.

classmethod SchemaLoaderWiki.**load**(*filename=None*, *schema_as_string=None*, *schema=None*, *file_format=None*, *name=''*)

Loads and returns the schema, including partnered schema if applicable.

Parameters

- **filename** (*str or None*) – A valid filepath or None
- **schema_as_string** (*str or None*) – A full schema as text or None
- **schema** (*HedSchema or None*) – A HED schema to merge this new file into It must be a with-standard schema with the same value.
- **file_format** (*str or None*) – If this is an owl file being loaded, this is the format. Allowed values include: turtle, json-ld, and owl(xml)
- **name** (*str or None*) – Optional user supplied identifier, by default uses filename

Returns
The new schema

Return type
schema(HedSchema)

static SchemaLoaderWiki.**parse_star_string**(*s*)

SchemaLoaderWiki.**schema**

The partially loaded schema if you are after just header attributes.

3.3.15.13 wiki_constants

Classes

HedWikiSection()

3.3.15.13.1 HedWikiSection

class HedWikiSection

Methods

HedWikiSection.__init__()

Attributes

HedWikiSection.Attributes
HedWikiSection.EndHed
HedWikiSection.EndSchema
HedWikiSection.Epilogue
HedWikiSection.ExternalAnnotations
HedWikiSection.HeaderLine
HedWikiSection.Prefixes
HedWikiSection.Prologue
HedWikiSection.Properties
HedWikiSection.Schema
HedWikiSection.Sources
HedWikiSection.UnitModifiers
HedWikiSection.UnitsClasses
HedWikiSection.ValueClasses

HedWikiSection.**__init__**()

HedWikiSection.**Attributes** = 9

HedWikiSection.**EndHed** = 15

HedWikiSection.**EndSchema** = 5

HedWikiSection.**Epilogue** = 11

HedWikiSection.**ExternalAnnotations** = 14

HedWikiSection.**HeaderLine** = 2

HedWikiSection.**Prefixes** = 13

HedWikiSection.**Prologue** = 3

HedWikiSection.**Properties** = 10

HedWikiSection.**Schema** = 4

HedWikiSection.**Sources** = 12

HedWikiSection.**UnitModifiers = 7**

HedWikiSection.**UnitsClasses = 6**

HedWikiSection.**ValueClasses = 8**

3.3.15.14 xml2schema

This module is used to create a HedSchema object from an XML file or tree.

Classes

SchemaLoaderXML(filename[, ...])	Loads XML schemas from filenames or strings.

3.3.15.14.1 SchemaLoaderXML

class SchemaLoaderXML(*filename*, *schema_as_string=None*, *schema=None*, *file_format=None*, *name=''*)

Loads XML schemas from filenames or strings.

Expected usage is SchemaLoaderXML.load(filename)

SchemaLoaderXML(filename) will load just the header_attributes

Methods

SchemaLoaderXML.__init__(filename[, ...])	Loads the given schema from one of the two parameters.
SchemaLoaderXML.find_rooted_entry(tag_entry, ...)	This semi-validates rooted tags, raising an exception on major errors
SchemaLoaderXML.fix_extra(key)	
SchemaLoaderXML.fix_extras()	Fixes the extras after loading the schema, to ensure they are in the correct format.
SchemaLoaderXML.load([filename, ...])	Loads and returns the schema, including partnered schema if applicable.

Attributes

SchemaLoaderXML.schema	The partially loaded schema if you are after just header attributes.

SchemaLoaderXML.**__init__**(*filename*, *schema_as_string=None*, *schema=None*, *file_format=None*, *name=''*)

Loads the given schema from one of the two parameters.

Parameters

- **filename** (*str or None*) – A valid filepath or None
- **schema_as_string** (*str or None*) – A full schema as text or None

- **schema** (*HedSchema or None*) – A HED schema to merge this new file into It must be a with-standard schema with the same value.
- **file_format** (*str or None*) – The format of this file if needed(only for owl currently)
- **name** (*str or None*) – Optional user supplied identifier, by default uses filename

static SchemaLoaderXML.**find_rooted_entry**(*tag_entry*, *schema*, *loading_merged*)

This semi-validates rooted tags, raising an exception on major errors

Parameters

- **tag_entry** (*HedTagEntry*) – the possibly rooted tag
- **schema** (*HedSchema*) – The schema being loaded
- **loading_merged** (*bool*) – If this schema was already merged before loading

Returns

The base tag entry from the standard schema
Returns None if this tag isn't rooted

Return type
rooted_tag(HedTagEntry or None)

Raises
HedFileError –

- A rooted attribute is found in a non-paired schema
- A rooted attribute is not a string
- A rooted attribute was found on a non-root node in an unmerged schema.
- A rooted attribute is found on a root node in a merged schema.
- A rooted attribute indicates a tag that doesn't exist in the base schema.

SchemaLoaderXML.**fix_extra**(*key*)

SchemaLoaderXML.**fix_extras**()

Fixes the extras after loading the schema, to ensure they are in the correct format.

classmethod SchemaLoaderXML.**load**(*filename=None*, *schema_as_string=None*, *schema=None*, *file_format=None*, *name=''*)

Loads and returns the schema, including partnered schema if applicable.

Parameters

- **filename** (*str or None*) – A valid filepath or None
- **schema_as_string** (*str or None*) – A full schema as text or None
- **schema** (*HedSchema or None*) – A HED schema to merge this new file into It must be a with-standard schema with the same value.
- **file_format** (*str or None*) – If this is an owl file being loaded, this is the format. Allowed values include: turtle, json-ld, and owl(xml)
- **name** (*str or None*) – Optional user supplied identifier, by default uses filename

Returns
The new schema

Return type
schema(HedSchema)

SchemaLoaderXML.**schema**
The partially loaded schema if you are after just header attributes.

3.3.15.15 xml_constants

Constants used for the

3.3.16 schema_validation_util

Utilities used in HED validation/loading using a HED schema.

Functions

get_allowed_characters(value_classes)	Returns the allowed characters in a given container of value classes
get_allowed_characters_by_name(...)	Returns the allowed characters from a list of character set names
get_problem_indexes(validation_string, ...)	Finds indexes with values not in character set
schema_version_for_library(hed_schema, ...)	Given the library name and HED schema object, return the version
validate_schema_description_new(hed_entry)	Check the description of the entry for invalid character issues
validate_schema_tag_new(hed_entry)	Check tag entry for capitalization and illegal characters.
validate_schema_term_new(hed_entry[, hed_term])	Check the term for invalid character issues

get_allowed_characters(*value_classes*)
Returns the allowed characters in a given container of value classes

Parameters
value_classes (*list of HedSchemaEntry*) – A list of schema entries that should have the allowedCharacter attribute

Returns
The set of all characters from the given classes

Return type
character_set(set)

get_allowed_characters_by_name(*character_set_names*)
Returns the allowed characters from a list of character set names

Note: "nonascii" is a special case "character" that can be included as well

Parameters
character_set_names (*list of str*) – A list of character sets to allow. See hed_schema_constants.character_types

Returns
The set of all characters from the names

Return type
character_set(set)

get_problem_indexes(*validation_string*, *character_set*, *index_adj=0*)

Finds indexes with values not in character set

Parameters

- **validation_string** (*str*) – The string to check characters in
- **character_set** (*set*) – the list of valid characters(or the value "nonascii" as a set entry)
- **index_adj** (*int*) – the value to adjust the reported indices by, if this isn't the start of a string.

Returns
The list of problematic characters and indices

Return type
index_list(tuple of (str, int))

schema_version_for_library(*hed_schema*, *library_name*)

Given the library name and HED schema object, return the version

Parameters

- **hed_schema** (*HedSchema*) – the schema object
- **library_name** (*str or None*) – The library name you're interested in. "" for the standard schema.

Returns
The version number of the given library name. Returns None if unknown library_name.

Return type
version_number (str)

validate_schema_description_new(*hed_entry*)

Check the description of the entry for invalid character issues

Parameters
hed_entry (*HedSchemaEntry*) – A single schema entry

Returns
A list of all invalid characters found in description. Each issue is a dictionary.

Return type
list

validate_schema_tag_new(*hed_entry*)

Check tag entry for capitalization and illegal characters.

Parameters
hed_entry (*HedTagEntry*) – A single tag entry

Returns
A list of all formatting issues found in the term. Each issue is a dictionary.

Return type
list

validate_schema_term_new(*hed_entry*, *hed_term=None*)

Check the term for invalid character issues

Parameters

- **hed_entry** (*HedSchemaEntry*) – A single schema entry
- **hed_term** (*str or None*) – Use instead of hed_entry.name if present.

Returns
A list of all formatting issues found in the term. Each issue is a dictionary.

Return type
list

3.3.17 schema_validation_util_deprecated

Legacy validation for terms and descriptions prior to 8.3.0.

Functions

validate_schema_description(hed_entry)	Check the description of a single schema entry.
validate_schema_tag(hed_entry)	Check short tag for capitalization and illegal characters.
verify_no_brackets(hed_entry)	Extremely basic check to block curly braces

validate_schema_description(*hed_entry*)

Check the description of a single schema entry.

Parameters
hed_entry (*HedSchemaEntry*) – A single schema entry

Returns
A list of all formatting issues found in the description.

Return type
list

validate_schema_tag(*hed_entry*)

Check short tag for capitalization and illegal characters.

Parameters
hed_entry (*HedTagEntry*) – A single HED term.

Returns
A list of all formatting issues found in the term. Each issue is a dictionary.

Return type
list

verify_no_brackets(*hed_entry*)

Extremely basic check to block curly braces

Parameters
hed_entry (*HedSchemaEntry*) – A single schema entry

Returns
A list of issues for invalid characters found in the name

Return type
list

3.4 tools

HED remodeling, analysis and summarization tools.

Modules

hed.tools.analysis	Basic analysis tools.
hed.tools.bids	Models for BIDS datasets and files.
hed.tools.remodeling	Remodeling tools for revising and summarizing tabular files.
hed.tools.util	Data and file handling utilities.
hed.tools.visualization	Visualization tools for HED.

3.4.1 analysis

Basic analysis tools.

Modules

hed.tools.analysis.annotation_util	Utilities to facilitate annotation of events in BIDS.
hed.tools.analysis.column_name_summary	Summarize the unique column names in a dataset.
hed.tools.analysis.event_manager	Manager of events of temporal extent.
hed.tools.analysis.file_dictionary	A file dictionary keyed by entity indices.
hed.tools.analysis.hed_tag_counts	Classes for managing counts of HED tags for columnar files.
hed.tools.analysis.hed_tag_manager	Manager for HED tags from a columnar file.
hed.tools.analysis.hed_type	Manager a type variable and its associated context.
hed.tools.analysis.hed_type_counts	Classes for managing counts of tags for one type tag such as Condition-variable or Task.
hed.tools.analysis.hed_type_defs	Manager for definitions associated with a type such as condition-variable.
hed.tools.analysis.hed_type_factors	Manager for factor information for a columnar file.
hed.tools.analysis.hed_type_manager	Manager for type factors and type definitions.
hed.tools.analysis.key_map	A map of column value keys into new column values.
hed.tools.analysis.sequence_map	A map of containing the number of times a particular sequence of values in a column of a columnar file.
hed.tools.analysis.tabular_summary	Summarize the contents of columnar files.
hed.tools.analysis.temporal_event	A single event process with starting and ending times.

3.4.1.1 annotation_util

Utilities to facilitate annotation of events in BIDS.

Functions

check_df_columns(df[, required_cols])	Return a list of the specified columns that are missing from a dataframe.
df_to_hed(dataframe[, description_tag])	Create sidecar-like dictionary from a 4-column dataframe.
extract_tags(hed_string, search_tag)	Extract all instances of specified tag from a tag_string.
generate_sidecar_entry(column_name[, ...])	Create a sidecar column dictionary for column.
hed_to_df(sidecar_dict[, col_names])	Return a 4-column dataframe of HED portions of sidecar.
merge_hed_dict(sidecar_dict, hed_dict)	Update a JSON sidecar based on the hed_dict values.
series_to_factor(series)	Convert a series to an integer factor list.
str_to_tabular(tsv_str[, sidecar])	Return a TabularInput a tsv string.
strs_to_hed_objs(hed_strings, hed_schema)	Returns a list of HedString objects from a list of strings.
strs_to_sidecar(sidecar_strings)	Return a Sidecar from a sidecar as string or as a list of sidecars as strings.
to_factor(data[, column])	Convert data to an integer factor list.
to_strlist(obj_list)	Return a list with the objects converted to string except for None elements.

check_df_columns(*df*, *required_cols=('column_name', 'column_value', 'description', 'HED')*)

Return a list of the specified columns that are missing from a dataframe.

Parameters

- **df** (*DataFrame*) – Spreadsheet to check the columns of.
- **required_cols** (*tuple*) – List of column names that must be present.

Returns
List of column names that are missing.

Return type
list

df_to_hed(*dataframe*, *description_tag=True*)

Create sidecar-like dictionary from a 4-column dataframe.

Parameters

- **dataframe** (*DataFrame*) – A four-column Pandas DataFrame with specific columns.
- **description_tag** (*bool*) – If True description tag is included.

Returns
A dictionary compatible with BIDS JSON tabular file that includes HED.

Return type
dict

Notes

- The DataFrame must have the columns with names: column_name, column_value, description, and HED.

extract_tags(*hed_string*, *search_tag*)

Extract all instances of specified tag from a tag_string.

Parameters

- **hed_string** (*str*) – Tag string from which to extract tag.
- **search_tag** (*str*) – HED tag to extract.

Returns

- str: Tag string without the tags.
- list: A list of the tags that were extracted, for example descriptions.

Return type
tuple

generate_sidecar_entry(*column_name*, *column_values=None*)

Create a sidecar column dictionary for column.

Parameters

- **column_name** (*str*) – Name of the column.
- **column_values** – List of column values.

hed_to_df(*sidecar_dict*, *col_names=None*)

Return a 4-column dataframe of HED portions of sidecar.

Parameters

- **sidecar_dict** (*dict*) – A dictionary conforming to BIDS JSON events sidecar format.
- **col_names** (*list, None*) – A list of the cols to include in the flattened sidecar.

Returns
Four-column spreadsheet representing HED portion of sidecar.

Return type
DataFrame

Notes

- The returned DataFrame has columns: column_name, column_value, description, and HED.

merge_hed_dict(*sidecar_dict*, *hed_dict*)

Update a JSON sidecar based on the hed_dict values.

Parameters

- **sidecar_dict** (*dict*) – Dictionary representation of a BIDS JSON sidecar.
- **hed_dict** (*dict*) – Dictionary derived from a dataframe representation of HED in sidecar.

series_to_factor(*series*)

Convert a series to an integer factor list.

Parameters
series (*Series*) –

Returns
list - contains 0's and 1's, empty, 'n/a' and np.nan are converted to 0.

str_to_tabular(*tsv_str*, *sidecar=None*)

Return a TabularInput a tsv string.

Parameters

- **tsv_str** (*str*) – A string representing a tabular input.
- **sidecar** – An optional Sidecar object.

strs_to_hed_objs(*hed_strings*, *hed_schema*)

Returns a list of HedString objects from a list of strings.

Parameters

- **hed_strings** (*string or list*) – String or strings representing HED annotations.
- **hed_schema** (*HedSchema or HedSchemaGroup*) – Schema version for the strings.

Returns
list of HedString objects or None.

Return type
list or None

strs_to_sidecar(*sidecar_strings*)

Return a Sidecar from a sidecar as string or as a list of sidecars as strings.

Parameters
sidecar_strings (*string or list*) – String or strings representing sidecars.

Returns
the merged sidecar from the list.

Return type
Sidecar or None

to_factor(*data*, *column=None*)

Convert data to an integer factor list.

Parameters

- **data** (*Series or DataFrame*) –
- **column** (*str*) – Optional column name if DataFrame (otherwise column 0).

Returns
list - contains 0's and 1's, empty, 'n/a' and np.nan are converted to 0.

to_strlist(*obj_list*)

Return a list with the objects converted to string except for None elements.

Parameters
obj_list (*list*) – A list of objects that are None or have a str method.

Returns
A list with the objects converted to strings – except None values are preserved.

Return type
list

3.4.1.2 column_name_summary

Summarize the unique column names in a dataset.

Classes

ColumnNameSummary([name])	Summarize the unique column names in a dataset.

3.4.1.2.1 ColumnNameSummary

class ColumnNameSummary(*name=''*)

Summarize the unique column names in a dataset.

Methods

ColumnNameSummary.__init__([name])	
ColumnNameSummary.get_summary([as_json])	Return summary as an object or in JSON.
ColumnNameSummary.update(name, columns)	Update the summary based on columns associated with a file.
ColumnNameSummary.update_headers(column_names)	Update the unique combinations of column names.

Attributes

ColumnNameSummary.**__init__**(*name=''*)

ColumnNameSummary.**get_summary**(*as_json=False*)

Return summary as an object or in JSON.

Parameters
as_json (*bool*) – If False (the default), return the underlying summary object, otherwise transform to JSON.

ColumnNameSummary.**update**(*name*, *columns*)

Update the summary based on columns associated with a file.

Parameters

- **name** (*str*) – File name associated with the columns.
- **columns** (*list*) – List of file names.

ColumnNameSummary.**update_headers**(*column_names*)

Update the unique combinations of column names.

Parameters

column_names (*list*) – List of column names to update.

3.4.1.3 event_manager

Manager of events of temporal extent.

Classes

EventManager(input_data, hed_schema[, ...])	Manager of events of temporal extent.

3.4.1.3.1 EventManager

class EventManager(*input_data*, *hed_schema*, *extra_defs=None*)

Manager of events of temporal extent.

Methods

EventManager.__init__(input_data, hed_schema)	Create an event manager for an events file.
EventManager.compress_strings(list_to_compress)	Compress a list of lists of strings into a single str with comma-separated elements.
EventManager.get_type_defs(types)	Return a list of definition names (lower case) that correspond to any of the specified types.
EventManager.str_list_to_hed(str_list)	Create a HedString object from a list of strings.
EventManager.unfold_context([remove_types])	Unfold the event information into a tuple based on context.

Attributes

EventManager.**__init__**(*input_data*, *hed_schema*, *extra_defs=None*)

Create an event manager for an events file. Manages events of temporal extent.

Parameters

- **input_data** (*TabularInput*) – Represents an events file with its sidecar.
- **hed_schema** (*HedSchema*) – HED schema used.
- **extra_defs** (*DefinitionDict*) – Extra definitions not included in the input_data information.

Raises

HedFileError –

- if there are any unmatched offsets.

Notes: Keeps the events of temporal extend by their starting index in events file. These events are separated from the rest of the annotations, which are contained in self.hed_strings.

static EventManager.**compress_strings**(*list_to_compress*)

Compress a list of lists of strings into a single str with comma-separated elements.

Parameters
list_to_compress (*list*) – List of lists of HED str to turn into a list of single HED strings.

Returns
List of same length as list_to_compress with each entry being a str.

Return type
list

EventManager.**get_type_defs**(*types*)

Return a list of definition names (lower case) that correspond to any of the specified types.

Parameters
types (*list or None*) – List of tags that are treated as types such as 'Condition-variable'

Returns
List of definition names (lower-case) that correspond to the specified types

Return type
list

EventManager.**str_list_to_hed**(*str_list*)

Create a HedString object from a list of strings.

Parameters
str_list (*list*) – A list of strings to be concatenated with commas and then converted.

Returns
The converted list.

Return type
HedString or None

EventManager.**unfold_context**(*remove_types=[]*)

Unfold the event information into a tuple based on context.

Parameters
remove_types (*list*) – List of types to remove.

Returns
list of str or HedString representing the information without the events of temporal extent. list of str or HedString or None representing the onsets of the events of temporal extent. list of str or HedString or None representing the ongoing context information.

If the

3.4.1.4 file_dictionary

A file dictionary keyed by entity indices.

Classes

FileDictionary(collection_name, file_list[, ...])	A file dictionary keyed by entity pair indices.

3.4.1.4.1 FileDictionary

class FileDictionary(*collection_name*, *file_list*, *key_indices=(0, 2)*, *separator='_'*)

A file dictionary keyed by entity pair indices.

Notes

- The entities are identified as 0, 1, ... depending on order in the base filename.
- The entity key-value pairs are assumed separated by '_' unless a separator is provided.

Methods

FileDictionary.__init__(collection_name, ...)	Create a dictionary with full paths as values.
FileDictionary.create_file_dict(file_list, ...)	Create new dict based on key indices.
FileDictionary.get_file_path(key)	Return file path corresponding to key.
FileDictionary.iter_files()	Iterator over the files in this dictionary.
FileDictionary.key_diffs(other_dict)	Return symmetric key difference with another dict.
FileDictionary.make_file_dict(file_list[, ...])	Return a dictionary of files using entity keys.
FileDictionary.make_key(key_string[, ...])	Create a key from specified entities.
FileDictionary.output_files([title, logger])	Return a string with the output of the list.

Attributes

FileDictionary.file_dict	Dictionary of path values in this dictionary.
FileDictionary.file_list	List of path values in this dictionary.
FileDictionary.key_list	Keys in this dictionary.
FileDictionary.name	Name of this dictionary.

FileDictionary.**__init__**(*collection_name*, *file_list*, *key_indices=(0, 2)*, *separator='_'*)

Create a dictionary with full paths as values.

Parameters

- **collection_name** (*str*) – Name of the file collection for reference.
- **file_list** (*list, None*) – List containing full paths of files of interest.
- **key_indices** (*tuple, None*) – List of order of key-value pieces to assemble for the key.
- **separator** (*str*) – Character used to separate pieces of key name.

Notes

- This dictionary is used for cross listing BIDS style files for different studies.
-

Examples

If key_indices is (0, 2), the key generated for /tmp/sub-001_task-FaceCheck_run-01_events.tsv is sub_001_run-01.

`FileDictionary.create_file_dict(file_list, key_indices, separator)`

Create new dict based on key indices.

Parameters

- **file_list** (*list*) – Paths of the files to include.
- **key_indices** (*tuple*) – A tuple of integers representing order of entities for key.
- **separator** (*str*) – The separator used between entities to form the key.

`FileDictionary.get_file_path(key)`

Return file path corresponding to key.

Parameters
key (*str*) – Key used to retrieve the file path.

Returns
File path.

Return type
str

`FileDictionary.iter_files()`

Iterator over the files in this dictionary.

Yields
- *str* – Key into the dictionary. - file: File path.

`FileDictionary.key_diffs(other_dict)`

Return symmetric key difference with another dict.

Parameters
other_dict (*FileDictionary*) –

Returns
The symmetric difference of the keys in this dictionary and the other one.

Return type
list

`static FileDictionary.make_file_dict(file_list, key_indices=(0, 2), separator='_')`

Return a dictionary of files using entity keys.

Parameters

- **file_list** (*list*) – Paths to files to use.
- **key_indices** (*tuple*) – Positions of entities to use for key.
- **separator** (*str*) – Separator character used to construct key.

Returns
Key is based on key indices and value is a full path.

Return type
dict

static FileDictionary.**make_key**(*key_string*, *indices=(0, 2)*, *separator='_'*)

Create a key from specified entities.

Parameters

- **key_string** (*str*) – The string from which to extract the key (usually a filename or path).
- **indices** (*tuple*) – Positions of entity pairs to use as key.
- **separator** (*str*) – Separator between entity pairs in the created key.

Returns
The created key.

Return type
str

FileDictionary.**output_files**(*title=None*, *logger=None*)

Return a string with the output of the list.

Parameters

- **title** (*None, str*) – Optional title.
- **logger** (*HedLogger*) – Optional HED logger for recording.

Returns
The dictionary in string form.

Return type
str

Notes

- The logger is updated if available.

FileDictionary.**file_dict**

Dictionary of path values in this dictionary.

FileDictionary.**file_list**

List of path values in this dictionary.

FileDictionary.**key_list**

Keys in this dictionary.

FileDictionary.**name**

Name of this dictionary.

3.4.1.5 hed_tag_counts

Classes for managing counts of HED tags for columnar files.

Classes

HedTagCount(hed_tag, file_name)	Counts for a particular HedTag in particular file.
HedTagCounts(name[, total_events])	Counts of HED tags for a group of columnar files.

3.4.1.5.1 HedTagCount

class HedTagCount(*hed_tag*, *file_name*)

Counts for a particular HedTag in particular file.

Methods

HedTagCount.__init__(hed_tag, file_name)	**param hed_tag** The HedTag to keep track of.
HedTagCount.get_empty()	
HedTagCount.get_info([verbose])	Return counts for this tag.
HedTagCount.get_summary()	Return a dictionary summary of the events and files for this tag.
HedTagCount.set_value(hed_tag)	Update the tag term value counts for a HedTag.

Attributes

HedTagCount.**__init__**(*hed_tag*, *file_name*)

Parameters

- **hed_tag** (*HedTag*) – The HedTag to keep track of.
- **file_name** (*str*) – Name of the file associated with the tag.

HedTagCount.**get_empty**()

HedTagCount.**get_info**(*verbose=False*)

Return counts for this tag.

Parameters

verbose (*bool*) – If False (the default) only number of files included, otherwise a list of files.

Returns

Keys are 'tag', 'events', and 'files'.

Return type

dict

HedTagCount.**get_summary**()

Return a dictionary summary of the events and files for this tag.

Returns
dictionary summary of events and files that contain this tag.

Return type
dict

HedTagCount.**set_value**(*hed_tag*)

Update the tag term value counts for a HedTag.

Parameters
hed_tag (*HedTag or None*) – Item to use to update the value counts.

3.4.1.5.2 HedTagCounts

class HedTagCounts(*name*, *total_events=0*)

Counts of HED tags for a group of columnar files.

Parameters

- **name** (*str*) – An identifier for these counts (usually the filename of the tabular file).
- **total_events** (*int*) – The total number of events in the columnar file.

Methods

HedTagCounts.__init__(name[, total_events])	
HedTagCounts.create_template(tags)	Creates a dictionary with keys based on list of keys in tags dictionary.
HedTagCounts.get_summary()	Return a summary object containing the tag count information of this summary.
HedTagCounts.merge_tag_dicts(other_dict)	Merge the information from another dictionary with this object's tag dictionary.
HedTagCounts.organize_tags(tag_template)	Organize tags into categories as specified by the tag_template.
HedTagCounts.update_tag_counts(...)	Update the tag counts based on a HedString object.

Attributes

HedTagCounts.**__init__**(*name*, *total_events=0*)

static HedTagCounts.**create_template**(*tags*)

Creates a dictionary with keys based on list of keys in tags dictionary.

Parameters
tags (*dict*) – dictionary of tags and key lists.

Returns
Dictionary with keys in key lists and values are empty lists.

Return type
dict

Note: This class is used to organize the results of the tags based on a template for display.

HedTagCounts.**get_summary**()

Return a summary object containing the tag count information of this summary.

Returns
Keys are 'name', 'files', 'total_events', and 'details'.

Return type
dict

HedTagCounts.**merge_tag_dicts**(*other_dict*)

Merge the information from another dictionary with this object's tag dictionary.

Parameters
other_dict (*dict*) – Dictionary of tag, HedTagCount to merge.

HedTagCounts.**organize_tags**(*tag_template*)

Organize tags into categories as specified by the tag_template.

Parameters
tag_template (*dict*) – A dictionary whose keys are titles and values are lists of HED tags (str).

Returns
Keys are tags (strings) and values are list of HedTagCount for items fitting template. list: HedTagCount objects corresponding to tags that don't fit the template.

Return type
dict

HedTagCounts.**update_tag_counts**(*hed_string_obj*, *file_name*)

Update the tag counts based on a HedString object.

Parameters

- **hed_string_obj** (*HedString*) – The HED string whose tags should be counted.
- **file_name** (*str*) – The name of the file corresponding to these counts.

3.4.1.6 hed_tag_manager

Manager for HED tags from a columnar file.

Classes

HedTagManager(event_manager[, remove_types, ...])	Manager for the HED tags from a columnar file.

3.4.1.6.1 HedTagManager

class HedTagManager(*event_manager*, *remove_types=[]*, *extra_defs=None*)

Manager for the HED tags from a columnar file.

Methods

HedTagManager.__init__(event_manager[, ...])	Create a tag manager for one tabular file.
HedTagManager.get_hed_obj(hed_str[, ...])	Return a HED string object with the types removed.
HedTagManager.get_hed_objs([...])	Return a list of HED string objects of same length as the tabular file.

Attributes

HedTagManager.**__init__**(*event_manager*, *remove_types=[]*, *extra_defs=None*)

Create a tag manager for one tabular file.

Parameters

- **event_manager** (*EventManager*) – an event manager for the tabular file.
- **remove_types** (*list or None*) – List of type tags (such as condition-variable) to remove.

HedTagManager.**get_hed_obj**(*hed_str*, *remove_types=False*, *remove_group=False*)

Return a HED string object with the types removed.

Parameters

- **hed_str** (*str*) – Represents a HED string.
- **remove_types** (*bool*) – If False (the default), do not remove the types managed by this manager.
- **remove_group** (*bool*) – If False (the default), do not remove the group when removing a type tag, otherwise remove its enclosing group.

HedTagManager.**get_hed_objs**(*include_context=True*, *replace_defs=False*)

Return a list of HED string objects of same length as the tabular file.

Parameters

- **include_context** (*bool*) – If True (default), include the Event-context group in the HED string.
- **replace_defs** (*bool*) – If True (default=False), replace the Def tags with Definition contents.

Returns

list - List of HED strings of same length as tabular file.

3.4.1.7 hed_type

Manager a type variable and its associated context.

Classes

HedType(event_manager, name[, type_tag])	Manager of a type variable and its associated context.

3.4.1.7.1 HedType

class HedType(*event_manager*, *name*, *type_tag='condition-variable'*)

Manager of a type variable and its associated context.

Methods

HedType.__init__(event_manager, name[, type_tag])	Create a variable manager for one type-variable for one tabular file.
HedType.get_summary()	
HedType.get_type_def_names()	Return the type defs names
HedType.get_type_factors([type_values, ...])	Create a dataframe with the indicated type tag values as factors.
HedType.get_type_list(type_tag, item)	Find a list of the given type tag from a HedTag, HedGroup, or HedString.
HedType.get_type_value_factors(type_value)	Return the HedTypeFactors associated with type_name or None.
HedType.get_type_value_level_info(type_value)	Return type variable corresponding to type_value.
HedType.get_type_value_names()	

Attributes

HedType.total_events	
HedType.type_variables	

HedType.**__init__**(*event_manager*, *name*, *type_tag='condition-variable'*)

Create a variable manager for one type-variable for one tabular file.

Parameters

- **event_manager** (*EventManager*) – Event manager instance
- **name** (*str*) – Name of the tabular file as a unique identifier.
- **type_tag** (*str*) – Lowercase short form of the tag to be managed.

Raises

HedFileError –

- On errors such as unmatched onsets or missing definitions.

HedType.**get_summary**()

HedType.**get_type_def_names**()

Return the type defs names

HedType.**get_type_factors**(*type_values=None*, *factor_encoding='one-hot'*)

Create a dataframe with the indicated type tag values as factors.

Parameters

- **type_values** (*list or None*) – A list of values of type tags for which to generate factors.
- **factor_encoding** (*str*) – Type of factor encoding (one-hot or categorical).

Returns

Contains the specified factors associated with this type tag.

Return type

DataFrame

static HedType.**get_type_list**(*type_tag*, *item*)

Find a list of the given type tag from a HedTag, HedGroup, or HedString.

Parameters

- **type_tag** (*str*) – a tag whose direct items you wish to remove
- **item** (*HedTag or HedGroup*) – The item from which to extract condition type_variables.

Returns

List of the items with this type_tag

Return type

list

HedType.**get_type_value_factors**(*type_value*)

Return the HedTypeFactors associated with type_name or None.

Parameters

type_value (*str*) – The tag corresponding to the type's value (such as the name of the condition variable).

Returns

HedTypeFactors or None

HedType.**get_type_value_level_info**(*type_value*)

Return type variable corresponding to type_value.

Parameters

type_value (*str*) –

Returns:

HedType.**get_type_value_names**()

HedType.**total_events**

HedType.**type_variables**

3.4.1.8 hed_type_counts

Classes for managing counts of tags for one type tag such as Condition-variable or Task.

Classes

HedTypeCount(type_value, type_tag[, file_name])	Manager of the counts of tags for one type tag such as Condition-variable or Task.
HedTypeCounts(name, type_tag)	Manager for summaries of tag counts for columnar files.

3.4.1.8.1 HedTypeCount

class HedTypeCount(*type_value*, *type_tag*, *file_name=None*)

Manager of the counts of tags for one type tag such as Condition-variable or Task.

Parameters

- **type_value** (*str*) – The value of the variable to be counted.
- **type_tag** (*str*) – The type of variable.

Examples

HedTypeCounts('SymmetricCond', 'condition-variable') keeps counts of Condition-variable/Symmetric.

Methods

HedTypeCount.__init__(type_value, type_tag)	

HedTypeCount.get_summary()	Return the summary of one value of one type tag.
HedTypeCount.to_dict()	Return count information as a dictionary.
HedTypeCount.update(type_sum, file_id)	Update the counts from a HedTypeValues.

Attributes

HedTypeCount.**__init__**(*type_value*, *type_tag*, *file_name=None*)

HedTypeCount.**get_summary**()

Return the summary of one value of one type tag.

Returns
Count information for one tag of one type.

Return type
dict

HedTypeCount.**to_dict**()

Return count information as a dictionary.

HedTypeCount.**update**(*type_sum*, *file_id*)

Update the counts from a HedTypeValues.

Parameters

- **type_sum** (*dict*) – Information about the contents for a particular data file.
- **file_id** (*str or None*) – Name of the file associated with the counts.

3.4.1.8.2 HedTypeCounts

class HedTypeCounts(*name*, *type_tag*)

Manager for summaries of tag counts for columnar files.

Methods

HedTypeCounts.__init__(name, type_tag)	

HedTypeCounts.add_descriptions(type_defs)	Update this summary based on the type variable map.
HedTypeCounts.get_summary()	Return the information in the manager as a dictionary.
HedTypeCounts.update(counts)	Update count information based on counts in another HedTypeCounts.
HedTypeCounts.update_summary(type_sum[, ...])	Update this summary based on the type variable map.

Attributes

HedTypeCounts.**__init__**(*name*, *type_tag*)

HedTypeCounts.**add_descriptions**(*type_defs*)

Update this summary based on the type variable map.

Parameters
type_defs (*HedTypeDefs*) – Contains the information about the value of a type.

HedTypeCounts.**get_summary**()

Return the information in the manager as a dictionary.

Returns
Dict with keys ‘name’, ‘type_tag’, ‘files’, ‘total_events’, and ‘details’.

Return type
dict

HedTypeCounts.**update**(*counts*)

Update count information based on counts in another HedTypeCounts.

Parameters
counts (*HedTypeCounts*) – Information to use in the update.

HedTypeCounts.**update_summary**(*type_sum*, *total_events=0*, *file_id=None*)

Update this summary based on the type variable map.

Parameters

- **type_sum** (*dict*) – Contains the information about the value of a type.
- **total_events** (*int*) – Total number of events processed.
- **file_id** (*str*) – Unique identifier for the associated file.

3.4.1.9 hed_type_defs

Manager for definitions associated with a type such as condition-variable.

Classes

HedTypeDefs(definitions[, type_tag])	Manager for definitions associated with a type such as condition-variable.

3.4.1.9.1 HedTypeDefs

class HedTypeDefs(*definitions*, *type_tag='condition-variable'*)

Manager for definitions associated with a type such as condition-variable.

Properties:

def_map (dict): keys are definition names, values are dict {type_values, description, tags}.

Example: A definition 'famous-face-cond' with contents:

'(Condition-variable/Face-type,Description/A face that should be recognized.,(Image,(Face,Famous)))'

would have type_values ['face_type']. All items are strings not objects.

Methods

HedTypeDefs.__init__(definitions[, type_tag])	Create a definition manager for a type of variable.
HedTypeDefs.extract_def_names(item[, no_value])	Return a list of Def values in item.
HedTypeDefs.get_type_values(item)	Return a list of type_tag values in item.
HedTypeDefs.split_name(name[, lowercase])	Split a name/# or name/x into name, x.

Attributes

HedTypeDefs.type_def_names	Return list of names of definition that have this type-variable.
HedTypeDefs.type_names	Return list of names of the type-variables associated with type definitions.

HedTypeDefs.**__init__**(*definitions*, *type_tag='condition-variable'*)

Create a definition manager for a type of variable.

Parameters

- **definitions** (*dict or DefinitionDict*) – A dictionary of DefinitionEntry objects.
- **type_tag** (*str*) – Lower-case HED tag string representing the type managed.

static HedTypeDefs.**extract_def_names**(*item*, *no_value=True*)

Return a list of Def values in item.

Parameters

- **item** (*HedTag, HedGroup, or HedString*) – An item containing a def tag.
- **no_value** (*bool*) – If True, strip off extra values after the definition name.

Returns
A list of definition names (as strings).

Return type
list

HedTypeDefs.**get_type_values**(*item*)

Return a list of type_tag values in item.

Parameters
item (*HedTag, HedGroup, or HedString*) – An item potentially containing def tags.

Returns
A list of the unique values associated with this type

Return type
list

static HedTypeDefs.**split_name**(*name*, *lowercase=True*)

Split a name/# or name/x into name, x.

Parameters

- **name** (*str*) – The extension or value portion of a tag.
- **lowercase** (*bool*) – If True (default), return values are converted to lowercase.

Returns
name of the definition. str: value of the definition if it has one.

Return type
str

HedTypeDefs.**type_def_names**

Return list of names of definition that have this type-variable.

Returns
definition names that have this type.

Return type
list

`HedTypeDefs.type_names`

Return list of names of the type-variables associated with type definitions.

Returns
type names associated with the type definitions

Return type
list

3.4.1.10 hed_type_factors

Manager for factor information for a columnar file.

Classes

`HedTypeFactors`(type_tag, type_value, ...)	Holds index of positions for a variable type for A columnar file.

3.4.1.10.1 HedTypeFactors

class HedTypeFactors(*type_tag*, *type_value*, *number_elements*)

Holds index of positions for a variable type for A columnar file.

Methods

HedTypeFactors.__init__(type_tag, ...)	Constructor for HedTypeFactors.
HedTypeFactors.get_factors([factor_encoding])	Return a DataFrame of factor vectors for this type factor.
HedTypeFactors.get_summary()	Return the summary of the type tag value as a dictionary.

Attributes

HedTypeFactors.ALLOWED_ENCODINGS

`HedTypeFactors.__init__`(*type_tag*, *type_value*, *number_elements*)

Constructor for HedTypeFactors.

Parameters

- **type_tag** (*str*) – Lowercase string corresponding to a HED tag which has a takes value child.
- **type_value** (*str*) – The value of the type summarized by this class.

- **number_elements** (*int*) – Number of elements in the data column

HedTypeFactors.**get_factors**(*factor_encoding='one-hot'*)

Return a DataFrame of factor vectors for this type factor.

Parameters
factor_encoding (*str*) – Specifies type of factor encoding (one-hot or categorical).

Returns
DataFrame containing the factor vectors as the columns.

Return type
DataFrame

HedTypeFactors.**get_summary**()

Return the summary of the type tag value as a dictionary.

Returns
Contains the summary.

Return type
dict

HedTypeFactors.**ALLOWED_ENCODINGS = ('categorical', 'one-hot')**

3.4.1.11 hed_type_manager

Manager for type factors and type definitions.

Classes

HedTypeManager(event_manager)	Manager for type factors and type definitions.

3.4.1.11.1 HedTypeManager

class HedTypeManager(*event_manager*)

Manager for type factors and type definitions.

Methods

HedTypeManager.__init__(event_manager)	Create a variable manager for one tabular file for all type variables.
HedTypeManager.add_type(type_name)	Add a type variable to be managed by this manager.
HedTypeManager.get_factor_vectors(type_tag)	Return a DataFrame of factor vectors for the indicated HED tag and values.
HedTypeManager.get_type(type_tag)	Returns the HedType variable associated with the type tag.
HedTypeManager.get_type_def_names(type_var)	Return the definitions associated with a particular type tag.
HedTypeManager.get_type_tag_factor(type_tag, ...)	Return the HedTypeFactors a specified value and extension.
HedTypeManager.summarize_all([as_json])	Return a dictionary containing the summaries for the types managed by this manager.

Attributes

HedTypeManager.types	Return a list of types managed by this manager.

HedTypeManager.**__init__**(*event_manager*)

Create a variable manager for one tabular file for all type variables.

Parameters
event_manager (*EventManager*) – An event manager for the tabular file.

Raises
HedFileError –

- On errors such as unmatched onsets or missing definitions.

HedTypeManager.**add_type**(*type_name*)

Add a type variable to be managed by this manager.

Parameters
type_name (*str*) – Type tag name of the type to be added.

HedTypeManager.**get_factor_vectors**(*type_tag*, *type_values=None*, *factor_encoding='one-hot'*)

Return a DataFrame of factor vectors for the indicated HED tag and values.

Parameters

- **type_tag** (*str*) – HED tag to retrieve factors for.
- **type_values** (*list or None*) – The values of the tag to create factors for or None if all unique values.
- **factor_encoding** (*str*) – Specifies type of factor encoding (one-hot or categorical).

Returns
DataFrame containing the factor vectors as the columns.

Return type
DataFrame or None

HedTypeManager.**get_type**(*type_tag*)

Returns the HedType variable associated with the type tag.

Parameters
type_tag (*str*) – HED tag to retrieve the type for.

Returns
the values associated with this type tag.

Return type
HedType or None

HedTypeManager.**get_type_def_names**(*type_var*)

Return the definitions associated with a particular type tag.

Parameters
type_var (*str*) – The name of a type tag such as Condition-variable.

Returns
Names of definitions that use this type.

Return type
list

HedTypeManager.**get_type_tag_factor**(*type_tag*, *type_value*)

Return the HedTypeFactors a specified value and extension.

Parameters

- **type_tag** (*str or None*) – HED tag for the type.
- **type_value** (*str or None*) – Value of this tag to return the factors for.

HedTypeManager.**summarize_all**(*as_json=False*)

Return a dictionary containing the summaries for the types managed by this manager.

Parameters
as_json (*bool*) – If False (the default), return as an object otherwise return as a JSON string.

Returns
Dictionary with the summary.

Return type
dict or str

HedTypeManager.**types**

Return a list of types managed by this manager.

Returns
Type tags names.

Return type
list

3.4.1.12 key_map

A map of column value keys into new column values.

Classes

KeyMap(key_cols[, target_cols, name])	A map of unique column values for remapping columns.

3.4.1.12.1 KeyMap

class KeyMap(*key_cols*, *target_cols=None*, *name=''*)

A map of unique column values for remapping columns.

key_cols

A list of column names that will be hashed into the keys for the map.

Type
list

target_cols

Optional list of column names that will be inserted into data and later remapped.

Type
list or None

name

An optional name of this remap for identification purposes.

Type
str

Notes: This mapping converts all columns in the mapping to strings. The remapping does not support other types of columns.

Methods

KeyMap.__init__(key_cols[, target_cols, name])	Information for remapping columns of tabular files.
KeyMap.make_template([additional_cols, ...])	Return a dataframe template.
KeyMap.remap(data)	Remap the columns of a dataframe or columnar file.
KeyMap.remove_quotes(df[, columns])	Remove quotes from the specified columns and convert to string.
KeyMap.resort()	Sort the col_map in place by the key columns.
KeyMap.update(data[, allow_missing])	Update the existing map with information from data.

Attributes

KeyMap.columns	Return the column names of the columns managed by this map.

KeyMap.**__init__**(*key_cols*, *target_cols=None*, *name=''*)

Information for remapping columns of tabular files.

Parameters

- **key_cols** (*list*) – List of columns to be replaced (assumed in the DataFrame).
- **target_cols** (*list*) – List of replacement columns (assumed to not be in the DataFrame).
- **name** (*str*) – Name associated with this remap (usually a pathname of the events file).

KeyMap.**make_template**(*additional_cols=None*, *show_counts=True*)

Return a dataframe template.

Parameters

- **additional_cols** (*list or None*) – Optional list of additional columns to append to the returned dataframe.
- **show_counts** (*bool*) – If True, number of times each key combination appears is in first column and values are sorted in descending order by.

Returns
A dataframe containing the template.

Return type
DataFrame

Raises
HedFileError –

- If additional columns are not disjoint from the key columns.

Notes

- The template consists of the unique key columns in this map plus additional columns.

KeyMap.**remap**(*data*)

Remap the columns of a dataframe or columnar file.

Parameters
data (*DataFrame, str*) – Columnar data (either DataFrame or filename) whose columns are to be remapped.

Returns

- DataFrame: New dataframe with columns remapped.
- list: List of row numbers that had no correspondence in the mapping.

Return type
tuple

Raises
HedFileError –

- If data is missing some of the key columns.

static KeyMap.**remove_quotes**(*df*, *columns=None*)

Remove quotes from the specified columns and convert to string.

Parameters

- **df** (*Dataframe*) – Dataframe to process by removing quotes.
- **columns** (*list*) – List of column names. If None, all columns are used.

Notes

- Replacement is done in place.

KeyMap.**resort**()

Sort the col_map in place by the key columns.

KeyMap.**update**(*data*, *allow_missing=True*)

Update the existing map with information from data.

Parameters

- **data** (*DataFrame or str*) – DataFrame or filename of an events file or event map.
- **allow_missing** (*bool*) – If True allow missing keys and add as n/a columns.

Raises
HedFileError –

- If there are missing keys and allow_missing is False.

KeyMap.**columns**

Return the column names of the columns managed by this map.

Returns
Column names of the columns managed by this map.

Return type
list

3.4.1.13 sequence_map

A map of containing the number of times a particular sequence of values in a column of a columnar file.

Classes

SequenceMap([codes, name])	A map of unique sequences of column values of a particular length appear in a columnar file.

3.4.1.13.1 SequenceMap

class SequenceMap(*codes=None*, *name=''*)

A map of unique sequences of column values of a particular length appear in a columnar file.

name

An optional name of this remap for identification purposes.

Type
str

Notes: This mapping converts all columns in the mapping to strings. The remapping does not support other types of columns.

Methods

SequenceMap.__init__([codes, name])	Information for setting up the maps.
SequenceMap.dot_str([group_spec])	Produce a DOT string representing this sequence map.
SequenceMap.edge_to_str(key)	Convert a graph edge to a DOT string.
SequenceMap.filter_edges()	
SequenceMap.get_edge_list([sort])	Return a DOT format edge list with the option of sorting by edge counts.
SequenceMap.prep(data)	Remove quotes from the specified columns and convert to string.
SequenceMap.update(data)	Update the existing map with information from data.

Attributes

SequenceMap.**__init__**(*codes=None*, *name=''*)

Information for setting up the maps.

Parameters

- **codes** (*list or None*) – If None use all codes, otherwise only include listed codes in the map.

- **name** (*str*) – Name associated with this remap (usually a pathname of the events file).

SequenceMap.**dot_str**(*group_spec={}*)

Produce a DOT string representing this sequence map.

SequenceMap.**edge_to_str**(*key*)

Convert a graph edge to a DOT string.

Parameters
key (*str*) – Hashcode string representing a graph edge.

SequenceMap.**filter_edges**()

SequenceMap.**get_edge_list**(*sort=True*)

Return a DOT format edge list with the option of sorting by edge counts.

Parameters
sort (*bool*) – If True (the default), the edge list is sorted by edge counts.

Returns
list of DOT strings representing the edges labeled by counts.

Return type
list

static SequenceMap.**prep**(*data*)

Remove quotes from the specified columns and convert to string.

Parameters
data (*Series*) – Dataframe to process by removing quotes.

Returns: Series .. rubric:: Notes

- Replacement is done in place.

SequenceMap.**update**(*data*)

Update the existing map with information from data.

Parameters

- **data** (*Series*) – DataFrame or filename of an events file or event map.
- **allow_missing** (*bool*) – If True allow missing keys and add as n/a columns.

Raises
HedFileError –

- If there are missing keys and allow_missing is False.

3.4.1.14 tabular_summary

Summarize the contents of columnar files.

Classes

TabularSummary([value_cols, skip_cols, name])	Summarize the contents of columnar files.

3.4.1.14.1 TabularSummary

class TabularSummary(*value_cols=None*, *skip_cols=None*, *name=''*)

Summarize the contents of columnar files.

Methods

TabularSummary.__init__([value_cols, ...])	Constructor for a BIDS tabular file summary.
TabularSummary.extract_sidecar_template()	Extract a BIDS sidecar-compatible dictionary.
TabularSummary.extract_summary(summary_info)	Create a TabularSummary object from a serialized summary.
TabularSummary.get_columns_info(dataframe[, ...])	Extract unique value counts for columns.
TabularSummary.get_number_unique([column_names])	Return the number of unique values in columns.
TabularSummary.get_summary([as_json])	Return the summary in dictionary format.
TabularSummary.make_combined_dicts(...[, ...])	Return combined and individual summaries.
TabularSummary.update(data[, name])	Update the counts based on data.
TabularSummary.update_summary(tab_sum)	Add TabularSummary values to this object.

Attributes

TabularSummary.**__init__**(*value_cols=None*, *skip_cols=None*, *name=''*)

Constructor for a BIDS tabular file summary.

Parameters

- **value_cols** (*list, None*) – List of columns to be treated as value columns.
- **skip_cols** (*list, None*) – List of columns to be skipped.
- **name** (*str*) – Name associated with the dictionary.

TabularSummary.**extract_sidecar_template**()

Extract a BIDS sidecar-compatible dictionary.

Returns
A sidecar template that can be converted to JSON.

Return type
dict

static TabularSummary.**extract_summary**(*summary_info*)

Create a TabularSummary object from a serialized summary.

Parameters
summary_info (*dict or str*) – A JSON string or a dictionary containing contents of a TabularSummary.

Returns
contains the information in summary_info as a TabularSummary object.

Return type
TabularSummary

static TabularSummary.**get_columns_info**(*dataframe*, *skip_cols=None*)

Extract unique value counts for columns.

Parameters

- **dataframe** (*DataFrame*) – The DataFrame to be analyzed.
- **skip_cols** (*list*) – List of names of columns to be skipped in the extraction.

Returns

A dictionary with keys that are column names and values that
are dictionaries of unique value counts.

Return type
dict

TabularSummary.**get_number_unique**(*column_names=None*)

Return the number of unique values in columns.

Parameters
column_names (*list, None*) – A list of column names to analyze or all columns if None.

Returns
Column names are the keys and the number of unique values in the column are the values.

Return type
dict

TabularSummary.**get_summary**(*as_json=False*)

Return the summary in dictionary format.

Parameters
as_json (*bool*) – If False, return as a Python dictionary, otherwise convert to a JSON dictionary.

static TabularSummary.**make_combined_dicts**(*file_dictionary*, *skip_cols=None*)

Return combined and individual summaries.

Parameters

- **file_dictionary** (*FileDictionary*) – Dictionary of file name keys and full path.
- **skip_cols** (*list*) – Name of the column.

Returns

- TabularSummary: Summary of the file dictionary.
- dict: of individual TabularSummary objects.

Return type
tuple

TabularSummary.**update**(*data*, *name=None*)

Update the counts based on data.

Parameters

- **data** (*DataFrame, str, or list*) – DataFrame containing data to update.

- **name** (*str*) – Name of the summary.

TabularSummary.**update_summary**(*tab_sum*)

Add TabularSummary values to this object.

Parameters

tab_sum (*TabularSummary*) – A TabularSummary to be combined.

Notes

- The value_cols and skip_cols are updated as long as they are not contradictory.
- A new skip column cannot be used.

3.4.1.15 temporal_event

A single event process with starting and ending times.

Classes

TemporalEvent(contents, start_index, start_time)	A single event process with starting and ending times.

3.4.1.15.1 TemporalEvent

class TemporalEvent(*contents*, *start_index*, *start_time*)

A single event process with starting and ending times.

Note: the contents have the Onset and duration removed.

Methods

TemporalEvent.__init__(contents, ...)	
TemporalEvent.set_end(end_index, end_time)	Set end time information for an event process.

Attributes

TemporalEvent.**__init__**(*contents*, *start_index*, *start_time*)

TemporalEvent.**set_end**(*end_index*, *end_time*)

Set end time information for an event process.

Parameters

- **end_index** (*int*) – Position of ending event marker corresponding to the end of this event process.
- **end_time** (*float*) – Ending time of the event (usually in seconds).

3.4.2 bids

Models for BIDS datasets and files.

Modules

hed.tools.bids.bids_dataset	The contents of a BIDS dataset.
hed.tools.bids.bids_file	Models a BIDS file.
hed.tools.bids.bids_file_group	A group of BIDS files with specified suffix name.
hed.tools.bids.bids_sidecar_file	Container for a BIDS sidecar file.
hed.tools.bids.bids_tabular_file	A BIDS tabular file including its associated sidecar.
hed.tools.bids.bids_util	

3.4.2.1 bids_dataset

The contents of a BIDS dataset.

Classes

`BidsDataset`(root_path[, schema, suffixes, ...])	A BIDS dataset representation primarily focused on HED evaluation.

3.4.2.1.1 BidsDataset

class **BidsDataset**(*root_path*, *schema=None*, *suffixes=['events', 'participants']*, *exclude_dirs=['sourcedata', 'derivatives', 'code', 'stimuli']*)

A BIDS dataset representation primarily focused on HED evaluation.

root_path

Real root path of the BIDS dataset.

Type
str

schema

The schema used for evaluation.

Type
HedSchema or HedSchemaGroup

file_groups

A dictionary of BidsFileGroup objects with a given file suffix.

Type
dict

Methods

BidsDataset.__init__(root_path[, schema, ...])	Constructor for a BIDS dataset.
BidsDataset.get_file_group(suffix)	Return the file group of files with the specified suffix.
BidsDataset.get_summary()	Return an abbreviated summary of the dataset.
BidsDataset.validate([check_for_warnings, ...])	Validate the dataset.

Attributes

BidsDataset.**__init__**(*root_path*, *schema=None*, *suffixes=['events', 'participants']*, *exclude_dirs=['sourcedata', 'derivatives', 'code', 'stimuli']*)

Constructor for a BIDS dataset.

Parameters

- **root_path** (*str*) – Root path of the BIDS dataset.
- **schema** (*HedSchema or HedSchemaGroup*) – A schema that overrides the one specified in dataset.
- **suffixes** (*list or None*) – File name suffixes of items to include. If None or empty, then ['_events', 'participants'] is assumed.
- **exclude_dirs=['sourcedata'** –
- **'derivatives'** –
- **'code'** –
- **'phenotype']** –

BidsDataset.**get_file_group**(*suffix*)

Return the file group of files with the specified suffix.

Parameters

suffix (*str*) – Suffix of the BidsFileGroup to be returned.

Returns

The requested tabular group.

Return type

BidsFileGroup or None

BidsDataset.**get_summary**()

Return an abbreviated summary of the dataset.

BidsDataset.**validate**(*check_for_warnings=False*, *schema=None*)

Validate the dataset.

Parameters

- **check_for_warnings** (*bool*) – If True, check for warnings.
- **schema** (*HedSchema or HedSchemaGroup or None*) – The schema used for validation.

Returns

List of issues encountered during validation. Each issue is a dictionary.

Return type
list

3.4.2.2 bids_file

Models a BIDS file.

Classes

BidsFile(file_path)	A BIDS file with entity dictionary.

3.4.2.2.1 BidsFile

class BidsFile(*file_path*)

A BIDS file with entity dictionary.

file_path

Real path of the file.

Type
str

suffix

Suffix part of the filename.

Type
str

ext

Extension (including the .).

Type
str

entity_dict

Dictionary of entity-names (keys) and entity-values (values).

Type
dict

Notes

- This class may hold the merged sidecar giving metadata for this file as well as contents.

Methods

BidsFile.__init__(file_path)	Constructor for a file path.
BidsFile.clear_contents()	Set the contents attribute of this object to None.
BidsFile.get_entity(entity_name)	Return the entity value for the specified entity.
BidsFile.get_key([entities])	Return a key for this BIDS file given a list of entities.
BidsFile.set_contents([content_info, overwrite])	Set the contents of this object.

Attributes

BidsFile.contents	Return the current contents of this object.

BidsFile.**__init__**(*file_path*)

Constructor for a file path.

Parameters
file_path (*str*) – Full path of the file.

BidsFile.**clear_contents**()

Set the contents attribute of this object to None.

BidsFile.**get_entity**(*entity_name*)

Return the entity value for the specified entity.

Parameters
entity_name (*str*) – Name of the BIDS entity, for example task, run, or sub.

Returns
Entity value if any, otherwise None.

Return type
str or None

BidsFile.**get_key**(*entities=None*)

Return a key for this BIDS file given a list of entities.

Parameters
entities (*tuple*) – A tuple of strings representing entities.

Returns
A key based on this object.

Return type
str

Notes

If entities is None, then the file path is used as the key.

BidsFile.**set_contents**(*content_info=None*, *overwrite=False*)

Set the contents of this object.

Parameters

- **content_info** (*Any*) – JSON dictionary The contents appropriate for this object.
- **overwrite** (*bool*) – If False and the contents are not empty, do nothing.

Notes

- Do not set if the contents are already set and no_overwrite is True.

`BidsFile.contents`
Return the current contents of this object.

3.4.2.3 bids_file_group

A group of BIDS files with specified suffix name.

Classes

`BidsFileGroup`(root_path, file_list[, suffix])	Container for BIDS files with a specified suffix.

3.4.2.3.1 BidsFileGroup

`class BidsFileGroup`(*root_path*, *file_list*, *suffix='events'*)
Container for BIDS files with a specified suffix.

`suffix`
The file suffix specifying the class of file represented in this group (e.g., events).

Type
str

`sidecar_dict`
A dictionary of sidecars associated with this suffix .

Type
dict

`datafile_dict`
A dictionary with values either BidsTabularFile or BidsTimeseriesFile.

Type
dict

`sidecar_dir_dict`
Dictionary whose keys are directory paths and values are list of sidecars in the corresponding directory.

Type
dict

Methods

BidsFileGroup.__init__(root_path, file_list)	Constructor for a BidsFileGroup.
BidsFileGroup.create_file_group(root_path, ...)	
BidsFileGroup.summarize([value_cols, skip_cols])	Return a BidsTabularSummary of group files.
BidsFileGroup.validate(hed_schema[, ...])	Validate the sidecars and datafiles and return a list of issues.
BidsFileGroup.validate_datafiles(hed_schema)	Validate the datafiles and return an error list.
BidsFileGroup.validate_sidecars(hed_schema)	Validate merged sidecars.

Attributes

BidsFileGroup.**__init__**(*root_path*, *file_list*, *suffix='events'*)

Constructor for a BidsFileGroup.

Parameters

- **file_list** (*list*) – List of paths to the relevant tsv and json files.
- **suffix** (*str*) – Suffix indicating the type this group represents (e.g. events, or channels, etc.).

static BidsFileGroup.**create_file_group**(*root_path*, *file_list*, *suffix*)

BidsFileGroup.**summarize**(*value_cols=None*, *skip_cols=None*)

Return a BidsTabularSummary of group files.

Parameters

- **value_cols** (*list*) – Column names designated as value columns.
- **skip_cols** (*list*) – Column names designated as columns to skip.

Returns
A summary of the number of values in different columns if tabular group.

Return type
TabularSummary or None

Notes

- The columns that are not value_cols or skip_col are summarized by counting

the number of times each unique value appears in that column.

BidsFileGroup.**validate**(*hed_schema*, *extra_def_dicts=None*, *check_for_warnings=False*)

Validate the sidecars and datafiles and return a list of issues.

Parameters

- **hed_schema** (*HedSchema*) – Schema to apply to the validation.
- **extra_def_dicts** (*DefinitionDict*) – Extra definitions that come from outside.
- **check_for_warnings** (*bool*) – If True, include warnings in the check.

Returns
A list of validation issues found. Each issue is a dictionary.

Return type
list

BidsFileGroup.**validate_datafiles**(*hed_schema*, *extra_def_dicts=None*, *error_handler=None*)

Validate the datafiles and return an error list.

Parameters

- **hed_schema** (*HedSchema*) – Schema to apply to the validation.
- **extra_def_dicts** (*DefinitionDict*) – Extra definitions that come from outside.
- **error_handler** (*ErrorHandler*) – Error handler to use.

Returns
A list of validation issues found. Each issue is a dictionary.

Return type
list

Notes: This will clear the contents of the datafiles if they were not previously set.

BidsFileGroup.**validate_sidecars**(*hed_schema*, *extra_def_dicts=None*, *error_handler=None*)

Validate merged sidecars.

Parameters

- **hed_schema** (*HedSchema*) – HED schema for validation.
- **extra_def_dicts** (*DefinitionDict*) – Extra definitions.
- **error_handler** (*ErrorHandler*) – Error handler to use.

Returns
A list of validation issues found. Each issue is a dictionary.

Return type
list

3.4.2.4 bids_sidecar_file

Container for a BIDS sidecar file.

Classes

BidsSidecarFile(file_path)	A BIDS sidecar file.

3.4.2.4.1 BidsSidecarFile

class BidsSidecarFile(*file_path*)

A BIDS sidecar file.

Methods

BidsSidecarFile.__init__(file_path)	Constructs a bids sidecar from a file.
BidsSidecarFile.clear_contents()	Set the contents attribute of this object to None.
BidsSidecarFile.get_entity(entity_name)	Return the entity value for the specified entity.
BidsSidecarFile.get_key([entities])	Return a key for this BIDS file given a list of entities.
BidsSidecarFile.is_hed(json_dict)	Return True if the json has HED.
BidsSidecarFile.is_sidecar_for(obj)	Return True if this is a sidecar for obj.
BidsSidecarFile.merge_sidecar_list(sidecar_list)	Merge a list of sidecars into a single sidecar.
BidsSidecarFile.set_contents([content_info, ...])	Set the contents of the sidecar.

Attributes

BidsSidecarFile.contents	Return the current contents of this object.

BidsSidecarFile.**__init__**(*file_path*)

Constructs a bids sidecar from a file.

Parameters
file_path (*str*) – The real path of the sidecar.

BidsSidecarFile.**clear_contents**()

Set the contents attribute of this object to None.

BidsSidecarFile.**get_entity**(*entity_name*)

Return the entity value for the specified entity.

Parameters
entity_name (*str*) – Name of the BIDS entity, for example task, run, or sub.

Returns
Entity value if any, otherwise None.

Return type
str or None

BidsSidecarFile.**get_key**(*entities=None*)

Return a key for this BIDS file given a list of entities.

Parameters
entities (*tuple*) – A tuple of strings representing entities.

Returns
A key based on this object.

Return type
str

Notes

If entities is None, then the file path is used as the key.

static BidsSidecarFile.**is_hed**(*json_dict*)

Return True if the json has HED.

Parameters
json_dict (*dict*) – A dictionary representing a JSON file or merged file.

Returns
True if the dictionary has HED or HED_assembled as a first or second-level key.

Return type
bool

BidsSidecarFile.**is_sidecar_for**(*obj*)

Return True if this is a sidecar for obj.

Parameters
obj (*BidsFile*) – A BidsFile object to check.

Returns
True if this is a BIDS parent of obj and False otherwise.

Return type
bool

Notes

- A sidecar is a sidecar for itself.

static BidsSidecarFile.**merge_sidecar_list**(*sidecar_list*, *name='merged_sidecar.json'*)

Merge a list of sidecars into a single sidecar.

Parameters

- **sidecar_list** (*list*) – A list of Sidecar objects.
- **name** (*str*) – The name of the merged sidecar.

Returns
A sidecar constructed from the merged list.

Return type
Sidecar or None

BidsSidecarFile.**set_contents**(*content_info=None*, *name='unknown'*, *overwrite=False*)

Set the contents of the sidecar.

Parameters

- **content_info** (*dict, or None*) – If None, create a Sidecar from the object's file-path.
- **name** (*str*) – The name of the sidecar.
- **overwrite** (*bool*) – If True, overwrite contents if already set.

Notes

- **The handling of content_info is as follows:**
  - None: This object's file_path is used.
  - dict: This is interpreted as a JSON dictionary.

BidsSidecarFile.**contents**

Return the current contents of this object.

3.4.2.5 bids_tabular_file

A BIDS tabular file including its associated sidecar.

Classes

BidsTabularFile(file_path)	A BIDS tabular file including its associated sidecar.

3.4.2.5.1 BidsTabularFile

class BidsTabularFile(*file_path*)

A BIDS tabular file including its associated sidecar.

Methods

BidsTabularFile.__init__(file_path)	Constructor for a BIDS tabular file.
BidsTabularFile.clear_contents()	Set the contents attribute of this object to None.
BidsTabularFile.get_entity(entity_name)	Return the entity value for the specified entity.
BidsTabularFile.get_key([entities])	Return a key for this BIDS file given a list of entities.
BidsTabularFile.set_contents([content_info, ...])	Set the contents of this tabular file (a TabularInput object).
BidsTabularFile.set_sidecar(sidecar)	Set the sidecar for this tabular file.

Attributes

BidsTabularFile.contents	Return the current contents of this object.

BidsTabularFile.**__init__**(*file_path*)

Constructor for a BIDS tabular file.

Parameters
file_path (*str*) – Path of the tabular file.

BidsTabularFile.**clear_contents**()

Set the contents attribute of this object to None.

BidsTabularFile.**get_entity**(*entity_name*)

Return the entity value for the specified entity.

Parameters
entity_name (*str*) – Name of the BIDS entity, for example task, run, or sub.

Returns
Entity value if any, otherwise None.

Return type
str or None

BidsTabularFile.**get_key**(*entities=None*)

Return a key for this BIDS file given a list of entities.

Parameters
entities (*tuple*) – A tuple of strings representing entities.

Returns
A key based on this object.

Return type
str

Notes

If entities is None, then the file path is used as the key.

BidsTabularFile.**set_contents**(*content_info=None*, *overwrite=False*)

Set the contents of this tabular file (a TabularInput object). It's sidecar should already be set.

Parameters

- **content_info** (*None*) – This always uses the internal file_path to create the contents.
- **overwrite** – If False (The Default), do not overwrite existing contents if any.

BidsTabularFile.**set_sidecar**(*sidecar*)

Set the sidecar for this tabular file.

Parameters
sidecar (*Sidecar*) – The sidecar for this tabular file.

BidsTabularFile.**contents**

Return the current contents of this object.

3.4.2.6 bids_util

Functions

get_candidates(source_dir, tsv_file_dict)	
get_merged_sidecar(root_path, tsv_file)	
get_schema_from_description(root_path)	
group_by_suffix(file_list)	Group files by suffix.
matches_criteria(json_file_dict, tsv_file_dict)	
parse_bids_filename(file_path)	Split a filename into BIDS-relevant components.
update_entity(name_dict, entity)	Update the dictionary with a new entity.
walk_back(root_path, file_path)	

get_candidates(*source_dir*, *tsv_file_dict*)

get_merged_sidecar(*root_path*, *tsv_file*)

get_schema_from_description(*root_path*)

group_by_suffix(*file_list*)

Group files by suffix.

Parameters
file_list (*list*) – List of file paths.

Returns
Dictionary with suffixes as keys and file lists as values.

Return type
dict

matches_criteria(*json_file_dict*, *tsv_file_dict*)

parse_bids_filename(*file_path*)

Split a filename into BIDS-relevant components.

Parameters
file_path (*str*) – Path to be parsed.

Returns
Dictionary with keys 'basename', 'suffix', 'prefix', 'ext', 'bad', and 'entities'.

Return type
dict

Notes

- Splits into BIDS suffix, extension, and a dictionary of entity name-value pairs.

update_entity(*name_dict*, *entity*)

Update the dictionary with a new entity.

Parameters

- **name_dict** (*dict*) – Dictionary of entities.
- **entity** (*str*) – Entity to be added.

walk_back(*root_path*, *file_path*)

3.4.3 remodeling

Remodeling tools for revising and summarizing tabular files.

Modules

hed.tools.remodeling.backup_manager	Manager for file backups for remodeling tools.
hed.tools.remodeling.cli	Command-line interface for remodeling tools.
hed.tools.remodeling.dispatcher	Controller for applying operations to tabular files and saving the results.
hed.tools.remodeling.operations	Remodeling operations.
hed.tools.remodeling.remodeler_validator	Validator for remodeler input files.

3.4.3.1 backup_manager

Manager for file backups for remodeling tools.

Classes

BackupManager(data_root[, backups_root])	Manager for file backups for remodeling tools.

3.4.3.1.1 BackupManager

class BackupManager(*data_root*, *backups_root=None*)

Manager for file backups for remodeling tools.

Methods

BackupManager.__init__(data_root[, backups_root])	Constructor for the backup manager.
BackupManager.create_backup(file_list[, ...])	Create a new backup from file_list.
BackupManager.get_backup(backup_name)	Return the dictionary corresponding to backup_name.
BackupManager.get_backup_files(backup_name)	Returns a list of full paths of files contained in the backup.
BackupManager.get_backup_path(backup_name, ...)	Retrieve the file from the backup or throw an error.
BackupManager.get_file_key(file_name)	
BackupManager.get_task(task_names, file_path)	Return the task if the file name contains a task_xxx where xxx is in task_names.
BackupManager.restore_backup([backup_name, ...])	Restore the files from backup_name to the main directory.

Attributes

BackupManager.BACKUP_DICTIONARY
BackupManager.BACKUP_ROOT
BackupManager.DEFAULT_BACKUP_NAME
BackupManager.RELATIVE_BACKUP_LOCATION

BackupManager.**__init__**(*data_root*, *backups_root=None*)

Constructor for the backup manager.

Parameters

- **data_root** (*str*) – Full path of the root of the data directory.
- **backups_root** (*str or None*) – Full path to the root where backups subdirectory is located.

Raises

HedFileError –

- If the data_root does not correspond to a real directory.

Notes: The backup_root will have remodeling/backups appended.

BackupManager.**create_backup**(*file_list*, *backup_name=None*, *verbose=False*)

Create a new backup from file_list.

Parameters

- **file_list** (*list*) – Full paths of the files to be in the backup.
- **backup_name** (*str or None*) – Name of the backup. If None, uses the default
- **verbose** (*bool*) – If True, print out the files that are being backed up.

Returns
True if the backup was successful. False if a backup of that name already exists.

Return type
bool

Raises

- *HedFileError* –
  - For missing or incorrect files.
- **OS-related error** –
  - OS-related error when file copying occurs.

BackupManager.**get_backup**(*backup_name*)

Return the dictionary corresponding to backup_name.

Parameters
backup_name (*str*) – Name of the backup to be retrieved.

Returns
The dictionary with the backup info.

Notes

The dictionary with backup information has keys that are the paths of the backed up files relative to the backup root. The values in this dictionary are the dates on which the particular file was backed up.

BackupManager.**get_backup_files**(*backup_name*, *original_paths=False*)

Returns a list of full paths of files contained in the backup.

Parameters

- **backup_name** (*str*) – Name of the backup.
- **original_paths** (*bool*) – If True return the original paths.

Returns
Full paths of the original files backed (original_paths=True) or the paths in the backup.

Return type
list

Raises
HedFileError –

- If not backup named backup_name exists.

BackupManager.**get_backup_path**(*backup_name*, *file_name*)

Retrieve the file from the backup or throw an error.

Parameters

- **backup_name** (*str*) – Name of the backup.
- **file_name** (*str*) – Full path of the file to be retrieved.

Returns
Full path of the corresponding file in the backup.

Return type
str

BackupManager.**get_file_key**(*file_name*)

static BackupManager.**get_task**(*task_names*, *file_path*)

Return the task if the file name contains a task_xxx where xxx is in task_names.

Parameters

- **task_names** (*list*) – List of task names (without the *task_* prefix).
- **file_path** (*str*) – Path of the filename to be tested.

Returns

the task name or '' if there is no task_xxx or xxx is not in task_names.

Return type

str

BackupManager.**restore_backup**(*backup_name='default_back'*, *task_names=[]*, *verbose=True*)

Restore the files from backup_name to the main directory.

Parameters

- **backup_name** (*str*) – Name of the backup to restore.
- **task_names** (*list*) – A list of task names to restore.
- **verbose** (*bool*) – If True, print out the file names being restored.

BackupManager.**BACKUP_DICTIONARY = 'backup_lock.json'**

BackupManager.**BACKUP_ROOT = 'backup_root'**

BackupManager.**DEFAULT_BACKUP_NAME = 'default_back'**

BackupManager.**RELATIVE_BACKUP_LOCATION = './derivatives/remodel/backups'**

3.4.3.2 cli

Command-line interface for remodeling tools.

Modules

hed.tools.remodeling.cli.run_remodel	Main command-line program for running the remodeling tools.
hed.tools.remodeling.cli.run_remodel_backup	Command-line program for creating a remodeler backup.
hed.tools.remodeling.cli.run_remodel_restore	Command-line program for restoring files from remodeler backup.

3.4.3.2.1 run_remodel

Main command-line program for running the remodeling tools.

Functions

get_parser()	Create a parser for the run_remodel command-line arguments.
get_sidecar(data_dir, tsv_path)	Get the sidecar for a file if it exists.
handle_backup(args)	Restore the backup if applicable.
main([arg_list])	The command-line program.
parse_arguments([arg_list])	Parse the command line arguments or arg_list if given.
parse_tasks(files, task_args)	Parse the tasks argument to get a task list.
run_ops(dispatch, args, tabular_files)	Run the remodeler on files of a specified form in a directory tree.

get_parser()

Create a parser for the run_remodel command-line arguments.

Returns
A parser for parsing the command line arguments.

Return type
argparse.ArgumentParser

get_sidecar(*data_dir*, *tsv_path*)

Get the sidecar for a file if it exists.

Parameters

- **data_dir** (*str*) – Full path of the data directory.
- **tsv_path** (*str*) – Full path of the file.

Returns
The Sidecar if it exists, otherwise None.

Return type
Sidecar or None

handle_backup(*args*)

Restore the backup if applicable.

Parameters
args (*obj*) – Parsed arguments as an object.

Returns
Backup name if there was a backup done.

Return type
str or None

main(*arg_list=None*)

The command-line program.

Parameters
arg_list (*list or None*) – Called with value None when called from the command line. Otherwise, called with the command-line parameters as an argument list.

Raises
HedFileError –

- if the data root directory does not exist.
- if the specified backup does not exist.

parse_arguments(*arg_list=None*)

Parse the command line arguments or arg_list if given.

Parameters
arg_list (*list*) – List of command line arguments as a list.

Returns
Argument object. List: A list of parsed operations (each operation is a dictionary).

Return type
Object

Raises
ValueError –

- If the operations were unable to be correctly parsed.

parse_tasks(*files*, *task_args*)

Parse the tasks argument to get a task list.

Parameters

- **files** (*list*) – List of full paths of files.
- **task_args** (*str or list*) – The argument values for the task parameter.

run_ops(*dispatch*, *args*, *tabular_files*)

Run the remodeler on files of a specified form in a directory tree.

Parameters

- **dispatch** (*Dispatcher*) – Controls the application of the operations and backup.
- **args** (*argparse.Namespace*) – Dictionary of arguments and their values.
- **tabular_files** (*list*) – List of files to include in this run.

3.4.3.2.2 run_remodel_backup

Command-line program for creating a remodeler backup.

Functions

get_parser()	Create a parser for the run_remodel_backup command-line arguments.
main([arg_list])	The command-line program for making a remodel backup.

get_parser()

Create a parser for the run_remodel_backup command-line arguments.

Returns
A parser for parsing the command line arguments.

Return type
argparse.ArgumentParser

main(*arg_list=None*)

The command-line program for making a remodel backup.

Parameters
arg_list (*list or None*) – Called with value None when called from the command line. Otherwise, called with the command-line parameters as an argument list.

Raises
HedFileError –

- If the specified backup already exists.

3.4.3.2.3 run_remodel_restore

Command-line program for restoring files from remodeler backup.

Functions

get_parser()	Create a parser for the run_remodel_restore command-line arguments.
main([arg_list])	The command-line program for restoring a remodel backup.

get_parser()

Create a parser for the run_remodel_restore command-line arguments.

Returns
A parser for parsing the command line arguments.

Return type
argparse.ArgumentParser

main(*arg_list=None*)

The command-line program for restoring a remodel backup.

Parameters
arg_list (*list or None*) – Called with value None when called from the command line. Otherwise, called with the command-line parameters as an argument list.

Raises
HedFileError –

- if the specified backup does not exist.

3.4.3.3 dispatcher

Controller for applying operations to tabular files and saving the results.

Classes

Dispatcher(operation_list[, data_root, ...])	Controller for applying operations to tabular files and saving the results.

3.4.3.3.1 Dispatcher

class Dispatcher(*operation_list*, *data_root=None*, *backup_name='default_back'*, *hed_versions=None*)

Controller for applying operations to tabular files and saving the results.

Methods

Dispatcher.__init__(operation_list[, ...])	Constructor for the dispatcher.
Dispatcher.errors_to_str(messages[, title, sep])	Return an error string representing error messages in a list.
Dispatcher.get_data_file(file_designator)	Get the correct data file give the file designator.
Dispatcher.get_schema(hed_versions)	Return the schema objects represented by the hed_versions.
Dispatcher.get_summaries([file_formats])	Return the summaries in a dictionary of strings suitable for saving or archiving.
Dispatcher.get_summary_save_dir()	Return the directory in which to save the summaries.
Dispatcher.parse_operations(operation_list)	Return a parsed a list of remodeler operations.
Dispatcher.post_proc_data(df)	Replace all nan entries with 'n/a' for BIDS compliance.
Dispatcher.prep_data(df)	Make a copy and replace all n/a entries in the data frame by np.nan for processing.
Dispatcher.run_operations(file_path[, ...])	Run the dispatcher operations on a file.
Dispatcher.save_summaries([save_formats, ...])	Save the summary files in the specified formats.

Attributes

Dispatcher.REMODELING_SUMMARY_PATH

Dispatcher.**__init__**(*operation_list*, *data_root=None*, *backup_name='default_back'*, *hed_versions=None*)

Constructor for the dispatcher.

Parameters

- **operation_list** (*list*) – List of valid unparsed operations.
- **data_root** (*str or None*) – Root directory for the dataset. If none, then backups are not made.
- **hed_versions** (*str, list, HedSchema, or HedSchemaGroup*) – The HED schema.

Raises

- *HedFileError* –
  - If the specified backup does not exist.
- **ValueError** –
  - If any of the operations cannot be parsed correctly.

static Dispatcher.**errors_to_str**(*messages*, *title=''*, *sep='\n'*)

Return an error string representing error messages in a list.

Parameters

- **messages** (*list*) – List of error dictionaries each representing a single error.
- **title** (*str*) – If provided the title is concatenated at the top.
- **sep** (*str*) – Character used between lines in concatenation.

Returns
Single string representing the messages.

Return type
str

Dispatcher.**get_data_file**(*file_designator*)

Get the correct data file give the file designator.

Parameters
file_designator (*str, DataFrame*) – A dataFrame or the full path of the dataframe in the original dataset.

Returns
DataFrame after reading the path.

Return type
DataFrame

Raises
HedFileError –

- If a valid file cannot be found.

Notes

- If a string is passed and there is a backup manager, the string must correspond to the full path of the file in the original dataset. In this case, the corresponding backup file is read and returned.
- If a string is passed and there is no backup manager, the data file corresponding to the file_designator is read and returned.
- If a Pandas DataFrame, return a copy.

static Dispatcher.**get_schema**(*hed_versions*)

Return the schema objects represented by the hed_versions.

Parameters
hed_versions (*str, list, HedSchema, HedSchemaGroup*) – If str, interpreted as a version number.

Returns
Objects loaded from the hed_versions specification.

Return type
HedSchema or HedSchemaGroup

Dispatcher.**get_summaries**(*file_formats=['.txt', '.json']*)

Return the summaries in a dictionary of strings suitable for saving or archiving.

Parameters
file_formats (*list*) – List of formats for the context files ('.json' and '.txt' are allowed).

Returns
A list of dictionaries of summaries keyed to filenames.

Return type
list

Dispatcher.**get_summary_save_dir**()

Return the directory in which to save the summaries.

Returns
the data_root + remodeling summary path

Return type
str

Raises
HedFileError –

- If this dispatcher does not have a data_root.

static Dispatcher.**parse_operations**(*operation_list*)

Return a parsed a list of remodeler operations.

Parameters
operation_list (*list*) – List of JSON remodeler operations.

Returns
List of Python objects containing parsed remodeler operations.

Return type
list

static Dispatcher.**post_proc_data**(*df*)

Replace all nan entries with 'n/a' for BIDS compliance.

Parameters
df (*DataFrame*) – The DataFrame to be processed.

Returns
DataFrame with the 'np.nan replaced by 'n/a'.

Return type
DataFrame

static Dispatcher.**prep_data**(*df*)

Make a copy and replace all n/a entries in the data frame by np.nan for processing.

Parameters
df (*DataFrame*) –

Dispatcher.**run_operations**(*file_path*, *sidecar=None*, *verbose=False*)

Run the dispatcher operations on a file.

Parameters

- **file_path** (*str or DataFrame*) – Full path of the file to be remodeled or a DataFrame.
- **sidecar** (*Sidecar or file-like*) – Only needed for HED operations.
- **verbose** (*bool*) – If True, print out progress reports.

Returns
The processed dataframe.

Return type
DataFrame

Dispatcher.**save_summaries**(*save_formats=['.json', '.txt']*, *individual_summaries='separate'*, *summary_dir=None*, *task_name=''*)

Save the summary files in the specified formats.

Parameters

- **save_formats** (*list*) – A list of formats [".txt", ."json"]
- **individual_summaries** (*str*) – "consolidated", "individual", or "none".
- **summary_dir** (*str or None*) – Directory for saving summaries.
- **task_name** (*str*) – Name of task if summaries separated by task or "" if not separated.

Notes

The summaries are saved in the dataset derivatives/remodeling folder if no save_dir is provided.

Notes

- "consolidated" means that the overall summary and summaries of individual files are in one summary file.
- "individual" means that the summaries of individual files are in separate files.
- "none" means that only the overall summary is produced.

Dispatcher.**REMODELING_SUMMARY_PATH = 'remodel/summaries'**

3.4.3.4 operations

Remodeling operations.

Modules

hed.tools.remodeling.operations.base_op	Base class for remodeling operations.
hed.tools.remodeling.operations.base_summary	Abstract base class for the contents of summary operations.
hed.tools.remodeling.operations.convert_columns_op	Convert the type of the specified columns of a tabular file.
hed.tools.remodeling.operations.factor_column_op	Append to tabular file columns of factors based on column values.
hed.tools.remodeling.operations.factor_hed_tags_op	Append columns of factors based on column values to a columnar file.
hed.tools.remodeling.operations.factor_hed_type_op	Append to columnar file the factors computed from type variables.
hed.tools.remodeling.operations.merge_consecutive_op	Merge consecutive rows of a columnar file with same column value.
hed.tools.remodeling.operations.number_groups_op	Implementation in progress.
hed.tools.remodeling.operations.number_rows_op	Implementation in progress.
hed.tools.remodeling.operations.remap_columns_op	Map values in m columns in a columnar file into a new combinations in n columns.
hed.tools.remodeling.operations.remove_columns_op	Remove columns from a columnar file.
hed.tools.remodeling.operations.remove_rows_op	Remove rows from a columnar file based on the values in a specified row.
hed.tools.remodeling.operations.rename_columns_op	Rename columns in a columnar file.
hed.tools.remodeling.operations.reorder_columns_op	Reorder columns in a columnar file.
hed.tools.remodeling.operations.split_rows_op	Split rows in a columnar file with onset and duration columns into multiple rows based on a specified column.
hed.tools.remodeling.operations.summarize_column_names_op	Summarize the column names in a collection of tabular files.
hed.tools.remodeling.operations.summarize_column_values_op	Summarize the values in the columns of a columnar file.
hed.tools.remodeling.operations.summarize_definitions_op	Summarize the type_defs in the dataset.
hed.tools.remodeling.operations.summarize_hed_tags_op	Summarize the HED tags in collection of tabular files.
hed.tools.remodeling.operations.summarize_hed_type_op	Summarize a HED type tag in a collection of tabular files.
hed.tools.remodeling.operations.summarize_hed_validation_op	Validate the HED tags in a dataset and report errors.
hed.tools.remodeling.operations.summarize_sidecar_from_events_op	Create a JSON sidecar from column values in a collection of tabular files.
hed.tools.remodeling.operations.valid_operations	The valid operations for the remodeling tools.

3.4.3.4.1 base_op

Base class for remodeling operations.

Classes

BaseOp(parameters)	Base class for operations.

3.4.3.4.1.1 BaseOp

class BaseOp(*parameters*)

Base class for operations. All remodeling operations should extend this class.

Methods

BaseOp.__init__(parameters)	Constructor for the BaseOp class.
BaseOp.do_op(dispatcher, df, name[, sidecar])	Base class method to be overridden by each operation.
BaseOp.validate_input_data(parameters)	Validates whether operation parameters meet op-specific criteria beyond that captured in json schema.

Attributes

BaseOp.NAME
BaseOp.PARAMS

BaseOp.**__init__**(*parameters*)

Constructor for the BaseOp class. Should be extended by operations.

Parameters

parameters (*dict*) – A dictionary specifying the appropriate parameters for the operation.

abstract BaseOp.**do_op**(*dispatcher*, *df*, *name*, *sidecar=None*)

Base class method to be overridden by each operation.

Parameters

- **dispatcher** (*Dispatcher*) – Manages the operation I/O.
- **df** (*DataFrame*) – The tabular file to be remodeled.
- **name** (*str*) – Unique identifier for the data – often the original file path.
- **sidecar** (*Sidecar or file-like*) – A JSON sidecar needed for HED operations.

abstract static BaseOp.**validate_input_data**(*parameters*)

Validates whether operation parameters meet op-specific criteria beyond that captured in json schema.

Example: A check to see whether two input arrays are the same length.

Notes: The minimum implementation should return an empty list to indicate no errors were found.
If additional validation is necessary, method should perform the validation and return a list with user-friendly error strings.

`BaseOp.NAME`

`BaseOp.PARAMS`

3.4.3.4.2 base_summary

Abstract base class for the contents of summary operations.

Classes

`BaseSummary`(sum_op)	Abstract base class for summary contents.

3.4.3.4.2.1 BaseSummary

class BaseSummary(*sum_op*)

Abstract base class for summary contents. Should not be instantiated.

Parameters
sum_op (*BaseOp*) – Operation corresponding to this summary.

Methods

BaseSummary.__init__(sum_op)	
BaseSummary.dump_summary(filename, summary)	
BaseSummary.get_details_dict(summary_info)	Return the summary-specific information.
BaseSummary.get_individual(summary_details)	Return a dictionary of the individual file summaries.
BaseSummary.get_summary([individual_summaries])	Return a summary dictionary with the information.
BaseSummary.get_summary_details([...])	Return a dictionary with the details for individual files and the overall dataset.
BaseSummary.get_text_summary([...])	Return a complete text summary by assembling the individual pieces.
BaseSummary.get_text_summary_details([...])	Return a text summary of the information represented by this summary.
BaseSummary.merge_all_info()	Return merged information.
BaseSummary.save(save_dir[, file_formats, ...])	Save the summaries using the format indicated.
BaseSummary.save_visualizations(save_dir[, ...])	Save summary visualizations, if any, using the format indicated.
BaseSummary.update_summary(summary_dict)	Method to update summary for a given tabular input.

Attributes

BaseSummary.DISPLAY_INDENT

BaseSummary.INDIVIDUAL_SUMMARIES_PATH

BaseSummary.**__init__**(*sum_op*)

static BaseSummary.**dump_summary**(*filename*, *summary*)

abstract BaseSummary.**get_details_dict**(*summary_info*)

Return the summary-specific information.

Parameters
summary_info (*object*) – Summary to return info from.

Returns
dictionary with the results.

Return type
dict

Notes

Abstract method be implemented by each individual summary.

Notes

The expected return value is a dictionary of the form:

{"Name": "", "Total events": 0, "Total files": 0, "Files": [], "Specifics": {}}"

BaseSummary.**get_individual**(*summary_details*, *separately=True*)

Return a dictionary of the individual file summaries.

Parameters

- **summary_details** (*dict*) – Dictionary of the individual file summaries.
- **separately** (*bool*) – If True (the default), each individual summary has a header for separate output.

BaseSummary.**get_summary**(*individual_summaries='separate'*)

Return a summary dictionary with the information.

Parameters
individual_summaries (*str*) – "separate", "consolidated", or "none"

Returns
dict - dictionary with "Dataset" and "Individual files" keys.

Notes: The individual_summaries value is processed as follows:

- "separate" individual summaries are to be in separate files.
- "consolidated" means that the individual summaries are in same file as overall summary.

- "none" means that only the overall summary is produced.

BaseSummary.**get_summary_details**(*include_individual=True*)

Return a dictionary with the details for individual files and the overall dataset.

Parameters
include_individual (*bool*) – If True, summaries for individual files are included.

Returns
dict - a dictionary with 'Dataset' and 'Individual files' keys.

Notes

- The 'Dataset' value is either a string or a dictionary with the overall summary.
- **The 'Individual files' value is dictionary whose keys are file names and values are** their corresponding summaries.

Users are expected to provide merge_all_info and get_details_dict functions to support this.

BaseSummary.**get_text_summary**(*individual_summaries='separate'*)

Return a complete text summary by assembling the individual pieces.

Parameters
individual_summaries (*str*) – One of the values "separate", "consolidated", or "none".

Returns
Complete text summary.

Return type
str

Notes: The options are:

- "none": Just has "Dataset" key.
- "consolidated" Has "Dataset" and "Individual files" keys with the values of each is a string.
- "separate" Has "Dataset" and "Individual files" keys. The values of "Individual files" is a dict.

BaseSummary.**get_text_summary_details**(*include_individual=True*)

Return a text summary of the information represented by this summary.

Parameters
include_individual (*bool*) – If True (the default), individual summaries are in "Individual files".

abstract BaseSummary.**merge_all_info**()

Return merged information.

Returns
Consolidated summary of information.

Return type
object

Notes

Abstract method be implemented by each individual summary.

BaseSummary.**save**(*save_dir*, *file_formats=['.txt']*, *individual_summaries='separate'*, *task_name=''*)

Save the summaries using the format indicated.

Parameters

- **save_dir** (*str*) – Name of the directory to save the summaries in.
- **file_formats** (*list*) – List of file formats to use for saving.
- **individual_summaries** (*str*) – Save one file or multiple files based on setting.
- **task_name** (*str*) – If this summary corresponds to files from a task, the task_name is used in filename.

BaseSummary.**save_visualizations**(*save_dir*, *file_formats=['.svg']*, *individual_summaries='separate'*, *task_name=''*)

Save summary visualizations, if any, using the format indicated.

Parameters

- **save_dir** (*str*) – Name of the directory to save the summaries in.
- **file_formats** (*list*) – List of file formats to use for saving.
- **individual_summaries** (*str*) – Save one file or multiple files based on setting.
- **task_name** (*str*) – If this summary corresponds to files from a task, the task_name is used in filename.

abstract BaseSummary.**update_summary**(*summary_dict*)

Method to update summary for a given tabular input.

Parameters
summary_dict (*dict*) –

BaseSummary.**DISPLAY_INDENT = ' '**

BaseSummary.**INDIVIDUAL_SUMMARIES_PATH = 'individual_summaries'**

3.4.3.4.3 convert_columns_op

Convert the type of the specified columns of a tabular file.

Classes

ConvertColumnsOp(parameters)	Convert specified columns to have specified data type.

3.4.3.4.3.1 ConvertColumnsOp

class ConvertColumnsOp(*parameters*)

Convert specified columns to have specified data type.

Required remodeling parameters:

- **column_names** (*list*): The list of columns to convert.
- **convert_to** (*str*): Name of type to convert to. (One of 'str', 'int', 'float', 'fixed'.)

Optional remodeling parameters:

- **decimal_places** (*int*): Number decimal places to keep (for fixed only).

Notes:

Methods

ConvertColumnsOp.__init__(parameters)	Constructor for the convert columns operation.
ConvertColumnsOp.do_op(dispatcher, df, name)	Convert the specified column to a specified type.
ConvertColumnsOp.validate_input_data(operations)	Additional validation required of operation parameters not performed by JSON schema validator.

Attributes

ConvertColumnsOp.NAME
ConvertColumnsOp.PARAMS

ConvertColumnsOp.**__init__**(*parameters*)

Constructor for the convert columns operation.

Parameters
parameters (*dict*) – Parameter values for required and optional parameters.

ConvertColumnsOp.**do_op**(*dispatcher*, *df*, *name*, *sidecar=None*)

Convert the specified column to a specified type.

Parameters

- **dispatcher** (*Dispatcher*) – Manages the operation I/O.
- **df** (*DataFrame*) – The DataFrame to be remodeled.
- **name** (*str*) – Unique identifier for the dataframe – often the original file path.
- **sidecar** (*Sidecar or file-like*) – Only needed for HED operations.

Returns
A new DataFrame with the factor columns appended.

Return type
DataFrame

static ConvertColumnsOp.**validate_input_data**(*operations*)

Additional validation required of operation parameters not performed by JSON schema validator.

ConvertColumnsOp.**NAME = 'convert_columns'**

ConvertColumnsOp.**PARAMS = {'additionalProperties': False, 'if': {'properties': {'convert_to': {'const': 'fixed'}}}, 'properties': {'column_names': {'description': 'List of names of the columns whose types are to be converted to the specified type.', 'items': {'type': 'string'}, 'minItems': 1, 'type': 'array', 'uniqueItems': True}, 'convert_to': {'description': 'Data type to convert the columns to.', 'enum': ['str', 'int', 'float', 'fixed'], 'type': 'string'}, 'decimal_places': {'description': 'The number of decimal points if converted to fixed.', 'type': 'integer'}}, 'required': ['column_names', 'convert_to'], 'then': {'required': ['decimal_places']}, 'type': 'object'}**

3.4.3.4.4 factor_column_op

Append to tabular file columns of factors based on column values.

Classes

FactorColumnOp(parameters)	Append to tabular file columns of factors based on column values.

3.4.3.4.4.1 FactorColumnOp

class FactorColumnOp(*parameters*)

Append to tabular file columns of factors based on column values.

Required remodeling parameters:

- **column_name** (*str*): The name of a column in the DataFrame to compute factors from.

Optional remodeling parameters

- **factor_names** (*list*): Names to use as the factor columns.
- **factor_values** (*list*): Values in the column column_name to create factors for.

Notes

- If no factor_values are provided, factors are computed for each of the unique values in column_name column.
- If factor_names are provided, then factor_values must also be provided and the two lists be the same size.

Methods

FactorColumnOp.__init__(parameters)	Constructor for the factor column operation.
FactorColumnOp.do_op(dispatcher, df, name[, ...])	Create factor columns based on values in a specified column.
FactorColumnOp.validate_input_data(parameters)	Check that factor_names and factor_values have same length if given.

Attributes

FactorColumnOp.NAME
FactorColumnOp.PARAMS

FactorColumnOp.**__init__**(*parameters*)

Constructor for the factor column operation.

Parameters

parameters (*dict*) – Parameter values for required and optional parameters.

FactorColumnOp.**do_op**(*dispatcher*, *df*, *name*, *sidecar=None*)

Create factor columns based on values in a specified column.

Parameters

- **dispatcher** (*Dispatcher*) – Manages the operation I/O.
- **df** (*DataFrame*) – The DataFrame to be remodeled.
- **name** (*str*) – Unique identifier for the dataframe – often the original file path.
- **sidecar** (*Sidecar or file-like*) – Not needed for this operation.

Returns

A new DataFrame with the factor columns appended.

Return type

DataFrame

static FactorColumnOp.**validate_input_data**(*parameters*)

Check that factor_names and factor_values have same length if given.

FactorColumnOp.**NAME = 'factor_column'**

FactorColumnOp.**PARAMS = {'additionalProperties': False, 'dependentRequired': {'factor_names': ['factor_values']}, 'properties': {'column_name': {'description': 'Name of the column for which to create one-hot factors for unique values.', 'type': 'string'}, 'factor_names': {'description': 'Names of the resulting factor columns. If given must be same length as factor_values', 'items': {'type': 'string'}, 'minItems': 1, 'type': 'array', 'uniqueItems': True}, 'factor_values': {'description': 'Specific unique column values to compute factors for (otherwise all unique values).', 'items': {'type': 'string'}, 'minItems': 1, 'type': 'array', 'uniqueItems': True}}, 'required': ['column_name'], 'type': 'object'}**

3.4.3.4.5 factor_hed_tags_op

Append columns of factors based on column values to a columnar file.

Classes

FactorHedTagsOp(parameters)	Append columns of factors based on column values to a columnar file.

3.4.3.4.5.1 FactorHedTagsOp

class FactorHedTagsOp(*parameters*)

Append columns of factors based on column values to a columnar file.

Required remodeling parameters:

- **queries** (*list*): Queries to be applied successively as filters.

Optional remodeling parameters:

- **expand_context** (*bool*): Expand the context if True.
- **query_names** (*list*): Column names for the query factors.
- **remove_types** (*list*): Structural HED tags to be removed (such as Condition-variable or Task).
- **expand_context** (*bool*): If true, expand the context based on Onset, Offset, and Duration.

Notes

- If query names are not provided, *query1*, *query2*, ... are used.
- If query names are provided, the list must have same list as the number of queries.
- When the context is expanded, the effect of events for temporal extent is accounted for.

Methods

FactorHedTagsOp.__init__(parameters)	Constructor for the factor HED tags operation.
FactorHedTagsOp.do_op(dispatcher, df, name)	Factor the column using HED tag queries.
FactorHedTagsOp.validate_input_data(parameters)	Parse and valid the queries and return issues in parsing queries, if any.

Attributes

FactorHedTagsOp.NAME

FactorHedTagsOp.PARAMS

FactorHedTagsOp.**__init__**(*parameters*)

Constructor for the factor HED tags operation.

Parameters
parameters (*dict*) – Actual values of the parameters for the operation.

FactorHedTagsOp.**do_op**(*dispatcher*, *df*, *name*, *sidecar=None*)

Factor the column using HED tag queries.

Parameters

- **dispatcher** (*Dispatcher*) – Manages the operation I/O.
- **df** (*DataFrame*) – The DataFrame to be remodeled.
- **name** (*str*) – Unique identifier for the dataframe – often the original file path.
- **sidecar** (*Sidecar or file-like*) – Only needed for HED operations.

Returns
A new dataframe after processing.

Return type
Dataframe

Raises
ValueError –

- If a name for a new query factor column is already a column.

static FactorHedTagsOp.**validate_input_data**(*parameters*)

Parse and valid the queries and return issues in parsing queries, if any.

Parameters
parameters (*dict*) – Dictionary representing the actual operation values.

Returns
List of issues in parsing queries.

Return type
list

FactorHedTagsOp.**NAME = 'factor_hed_tags'**

FactorHedTagsOp.**PARAMS** = {'additionalProperties': False, 'properties': {'expand_context': {'description': 'If true, the assembled HED tags include the effects of temporal extent (e.g., Onset).', 'type': 'boolean'}, 'queries': {'description': 'List of HED tag queries to compute one-hot factors for.', 'items': {'type': 'string'}, 'minItems': 1, 'type': 'array', 'uniqueItems': True}, 'query_names': {'description': 'Optional column names for the queries.', 'items': {'type': 'string'}, 'minItems': 1, 'type': 'array', 'uniqueItems': True}, 'remove_types': {'descriptions': 'List of type tags to remove from before querying (e.g., Condition-variable, Task).', 'items': {'type': 'string'}, 'minItems': 1, 'type': 'array', 'uniqueItems': True}, 'replace_defs': {'description': 'If true, Def tags are replaced with definition contents.', 'type': 'boolean'}}, 'required': ['queries'], 'type': 'object'}

3.4.3.4.6 factor_hed_type_op

Append to columnar file the factors computed from type variables.

Classes

FactorHedTypeOp(parameters)	Append to columnar file the factors computed from type variables.

3.4.3.4.6.1 FactorHedTypeOp

class FactorHedTypeOp(*parameters*)

Append to columnar file the factors computed from type variables.

Required remodeling parameters:

- **type_tag** (*str*): HED tag used to find the factors (most commonly *condition-variable*).

Optional remodeling parameters:

- **type_values** (*list*): If provided, specifies which factor values to include.

Methods

FactorHedTypeOp.__init__(parameters)	Constructor for the factor HED type operation.
FactorHedTypeOp.do_op(dispatcher, df, name)	Factor columns based on HED type and append to tabular data.
FactorHedTypeOp.validate_input_data(parameters)	Additional validation required of operation parameters not performed by JSON schema validator.

Attributes

FactorHedTypeOp.NAME

FactorHedTypeOp.PARAMS

FactorHedTypeOp.**__init__**(*parameters*)

Constructor for the factor HED type operation.

Parameters
parameters (*dict*) – Actual values of the parameters for the operation.

FactorHedTypeOp.**do_op**(*dispatcher*, *df*, *name*, *sidecar=None*)

Factor columns based on HED type and append to tabular data.

Parameters

- **dispatcher** (*Dispatcher*) – Manages the operation I/O.
- **df** (*DataFrame*) – The DataFrame to be remodeled.
- **name** (*str*) – Unique identifier for the dataframe – often the original file path.
- **sidecar** (*Sidecar or file-like*) – Only needed for HED operations.

Returns
A new DataFame with that includes the factors.

Return type
DataFrame

Notes

- If column_name is not a column in df, df is just returned.

static FactorHedTypeOp.**validate_input_data**(*parameters*)

Additional validation required of operation parameters not performed by JSON schema validator.

FactorHedTypeOp.**NAME** = **'factor_hed_type'**

FactorHedTypeOp.**PARAMS** = **{'additionalProperties': False, 'properties': {'type_tag': {'description': 'Type tag to use for computing factor vectors (e.g., Condition-variable or Task).', 'type': 'string'}, 'type_values': {'description': 'If provided, only compute one-hot factors for these values of the type tag.', 'items': {'type': 'string'}, 'minItems': 1, 'type': 'array', 'uniqueItems': True}}, 'required': ['type_tag'], 'type': 'object'}**

3.4.3.4.7 merge_consecutive_op

Merge consecutive rows of a columnar file with same column value.

Classes

MergeConsecutiveOp(parameters)	Merge consecutive rows of a columnar file with same column value.

3.4.3.4.7.1 MergeConsecutiveOp

class MergeConsecutiveOp(*parameters*)

Merge consecutive rows of a columnar file with same column value.

Required remodeling parameters:

- **column_name** (*str*): name of column whose consecutive values are to be compared (the merge column).
- **event_code** (*str* or *int* or *float*): the particular value in the match column to be merged.
- **set_durations** (*bool*): If true, set the duration of the merged event to the extent of the merged events.
- **ignore_missing** (*bool*): If true, missing match_columns are ignored.

Optional remodeling parameters:

- **match_columns** (*list*): A list of columns whose values have to be matched for two events to be the same.

Notes

This operation is meant for time-based tabular files that have an onset column.

Methods

MergeConsecutiveOp.__init__(parameters)	Constructor for the merge consecutive operation.
MergeConsecutiveOp.do_op(dispatcher, df, name)	Merge consecutive rows with the same column value.
MergeConsecutiveOp.validate_input_data(...)	Verify that the column name is not in match columns.

Attributes

MergeConsecutiveOp.NAME
MergeConsecutiveOp.PARAMS

MergeConsecutiveOp.**__init__**(*parameters*)

Constructor for the merge consecutive operation.

Parameters
parameters (*dict*) – Actual values of the parameters for the operation.

MergeConsecutiveOp.**do_op**(*dispatcher*, *df*, *name*, *sidecar=None*)

Merge consecutive rows with the same column value.

Parameters

- **dispatcher** (*Dispatcher*) – Manages the operation I/O.
- **df** (*DataFrame*) – The DataFrame to be remodeled.
- **name** (*str*) – Unique identifier for the dataframe – often the original file path.
- **sidecar** (*Sidecar or file-like*) – Not needed for this operation.

Returns
A new dataframe after processing.

Return type
Dataframe

Raises
ValueError –

- If dataframe does not have the anchor column and ignore_missing is False.
- If a match column is missing and ignore_missing is False.
- If the durations were to be set and the dataframe did not have an onset column.
- If the durations were to be set and the dataframe did not have a duration column.

static MergeConsecutiveOp.**validate_input_data**(*parameters*)

Verify that the column name is not in match columns.

Parameters
parameters (*dict*) – Dictionary of parameters of actual implementation.

MergeConsecutiveOp.**NAME = 'merge_consecutive'**

MergeConsecutiveOp.**PARAMS = {'additionalProperties': False, 'properties': {'column_name': {'description': 'The name of the column to check for repeated consecutive codes.', 'type': 'string'}, 'event_code': {'description': 'The event code to match for duplicates.', 'type': ['string', 'number']}, 'ignore_missing': {'description': 'If true, missing match columns are ignored.', 'type': 'boolean'}, 'match_columns': {'description': 'List of columns whose values must also match to be considered a repeat.', 'items': {'type': 'string'}, 'type': 'array'}, 'set_durations': {'description': 'If true, then the duration should be computed based on start of first to end of last.', 'type': 'boolean'}}, 'required': ['column_name', 'event_code', 'set_durations', 'ignore_missing'], 'type': 'object'}**

3.4.3.4.8 number_groups_op

Implementation in progress.

Classes

NumberGroupsOp(parameters)	Implementation in progress.

3.4.3.4.8.1 NumberGroupsOp

class NumberGroupsOp(*parameters*)

Implementation in progress.

Methods

NumberGroupsOp.__init__(parameters)	Constructor for the BaseOp class.
NumberGroupsOp.do_op(dispatcher, df, name[, ...])	Add numbers to groups of events in dataframe.
NumberGroupsOp.validate_input_data(parameters)	Additional validation required of operation parameters not performed by JSON schema validator.

Attributes

NumberGroupsOp.NAME
NumberGroupsOp.PARAMS

NumberGroupsOp.**__init__**(*parameters*)

Constructor for the BaseOp class. Should be extended by operations.

Parameters

parameters (*dict*) – A dictionary specifying the appropriate parameters for the operation.

NumberGroupsOp.**do_op**(*dispatcher*, *df*, *name*, *sidecar=None*)

Add numbers to groups of events in dataframe.

Parameters

- **dispatcher** (*Dispatcher*) – Manages the operation I/O.
- **df** (*DataFrame*) – The DataFrame to be remodeled.
- **name** (*str*) – Unique identifier for the dataframe – often the original file path.
- **sidecar** (*Sidecar or file-like*) – Only needed for HED operations.

Returns

Dataframe - a new dataframe after processing.

static NumberGroupsOp.**validate_input_data**(*parameters*)

Additional validation required of operation parameters not performed by JSON schema validator.

NumberGroupsOp.**NAME = 'number_groups'**

NumberGroupsOp.**PARAMS = {'additionalProperties': False, 'properties': {'number_column_name': {'type': 'string'}, 'overwrite': {'type': 'boolean'}, 'source_column': {'type': 'string'}, 'start': {'additionalProperties': False, 'properties': {'inclusion': {'enum': ['include', 'exclude'], 'type': 'string'}, 'values': {'type': 'array'}}, 'required': ['values', 'inclusion'], 'type': 'object'}, 'stop': {'additionalProperties': False, 'properties': {'inclusion': {'enum': ['include', 'exclude'], 'type': 'string'}, 'values': {'type': 'array'}}, 'required': ['values', 'inclusion'], 'type': 'object'}}, 'required': ['number_column_name', 'source_column', 'start', 'stop'], 'type': 'object'}**

3.4.3.4.9 number_rows_op

Implementation in progress.

Classes

NumberRowsOp(parameters)	Implementation in progress.

3.4.3.4.9.1 NumberRowsOp

class NumberRowsOp(*parameters*)

Implementation in progress.

Methods

NumberRowsOp.__init__(parameters)	Constructor for the BaseOp class.
NumberRowsOp.do_op(dispatcher, df, name[, ...])	Add numbers events dataframe.
NumberRowsOp.validate_input_data(parameters)	Additional validation required of operation parameters not performed by JSON schema validator.

Attributes

NumberRowsOp.NAME
NumberRowsOp.PARAMS

NumberRowsOp.**__init__**(*parameters*)

Constructor for the BaseOp class. Should be extended by operations.

Parameters

parameters (*dict*) – A dictionary specifying the appropriate parameters for the operation.

NumberRowsOp.**do_op**(*dispatcher*, *df*, *name*, *sidecar=None*)

Add numbers events dataframe.

Parameters

- **dispatcher** (*Dispatcher*) – Manages operation I/O.
- **df** (*DataFrame*) –
  - The DataFrame to be remodeled.
- **name** (*str*) –
  - Unique identifier for the dataframe – often the original file path.
- **sidecar** (*Sidecar or file-like*) – Only needed for HED operations.

Returns

Dataframe - a new dataframe after processing.

static NumberRowsOp.**validate_input_data**(*parameters*)

Additional validation required of operation parameters not performed by JSON schema validator.

NumberRowsOp.**NAME = 'number_rows'**

NumberRowsOp.**PARAMS = {'additionalProperties': False, 'properties': {'match_value': {'additionalProperties': False, 'properties': {'column': {'type': 'string'}, 'value': {'type': ['string', 'number']}}, 'required': ['column', 'value'], 'type': 'object'}, 'number_column_name': {'type': 'string'}, 'overwrite': {'type': 'boolean'}}, 'required': ['number_column_name'], 'type': 'object'}**

3.4.3.4.10 remap_columns_op

Map values in m columns in a columnar file into a new combinations in n columns.

Classes

RemapColumnsOp(parameters)	Map values in m columns in a columnar file into a new combinations in n columns.

3.4.3.4.10.1 RemapColumnsOp

class RemapColumnsOp(*parameters*)

Map values in m columns in a columnar file into a new combinations in n columns.

Required remodeling parameters:

- **source_columns** (*list*): The key columns to map (m key columns).
- **destination_columns** (*list*): The destination columns to have the mapped values (n destination columns).
- **map_list** (*list*): A list of lists with the mapping.
- **ignore_missing** (*bool*): If True, entries whose key column values are not in map_list are ignored.

Optional remodeling parameters:

integer_sources (*list*): Source columns that should be treated as integers rather than strings.

Notes

Each list element list is of length m + n with the key columns followed by mapped columns.

TODO: Allow wildcards

Methods

RemapColumnsOp.__init__(parameters)	Constructor for the remap columns operation.
RemapColumnsOp.do_op(dispatcher, df, name[, ...])	Remap new columns from combinations of others.
RemapColumnsOp.validate_input_data(parameters)	Validates whether operation parameters meet op-specific criteria beyond that captured in json schema.

Attributes

RemapColumnsOp.NAME
RemapColumnsOp.PARAMS

RemapColumnsOp.**__init__**(*parameters*)

Constructor for the remap columns operation.

Parameters
parameters (*dict*) – Parameter values for required and optional parameters.

RemapColumnsOp.**do_op**(*dispatcher*, *df*, *name*, *sidecar=None*)

Remap new columns from combinations of others.

Parameters

- **dispatcher** (*Dispatcher*) – Manages the operation I/O.
- **df** (*DataFrame*) – The DataFrame to be remodeled.
- **name** (*str*) – Unique identifier for the dataframe – often the original file path.
- **sidecar** (*Sidecar or file-like*) – Not needed for this operation.

Returns
A new dataframe after processing.

Return type
Dataframe

Raises
ValueError –

- If ignore_missing is False and source values from the data are not in the map.

static RemapColumnsOp.**validate_input_data**(*parameters*)

Validates whether operation parameters meet op-specific criteria beyond that captured in json schema.

Example: A check to see whether two input arrays are the same length.

Notes: The minimum implementation should return an empty list to indicate no errors were found.
If additional validation is necessary, method should perform the validation and return a list with user-friendly error strings.

RemapColumnsOp.**NAME = 'remap_columns'**

RemapColumnsOp.**PARAMS = {'additionalProperties': False, 'properties': {'destination_columns': {'description': 'The columns to insert new values based on a key lookup of the source columns.', 'items': {'type': 'string'}, 'minItems': 1, 'type': 'array'}, 'ignore_missing': {'description': 'If true, insert missing source columns in the result, filled with n/a, else error.', 'type': 'boolean'}, 'integer_sources': {'description': 'A list of source column names whose values are to be treated as integers.', 'items': {'type': 'string'}, 'minItems': 1, 'type': 'array', 'uniqueItems': True}, 'map_list': {'description': 'An array of k lists each with m+n entries corresponding to the k unique keys.', 'items': {'items': {'type': ['string', 'number']}, 'minItems': 1, 'type': 'array'}, 'minItems': 1, 'type': 'array', 'uniqueItems': True}, 'source_columns': {'description': 'The columns whose values are combined to provide the remap keys.', 'items': {'type': 'string'}, 'minItems': 1, 'type': 'array'}}, 'required': ['source_columns', 'destination_columns', 'map_list', 'ignore_missing'], 'type': 'object'}**

3.4.3.4.11 remove_columns_op

Remove columns from a columnar file.

Classes

RemoveColumnsOp(parameters)	Remove columns from a columnar file.

3.4.3.4.11.1 RemoveColumnsOp

class RemoveColumnsOp(*parameters*)

Remove columns from a columnar file.

Required remodeling parameters:

- **column_names** (*list*): The names of the columns to be removed.
- **ignore_missing** (*boolean*): If True, names in column_names that are not columns in df should be ignored.

Methods

RemoveColumnsOp.__init__(parameters)	Constructor for remove columns operation.
RemoveColumnsOp.do_op(dispatcher, df, name)	Remove indicated columns from a dataframe.
RemoveColumnsOp.validate_input_data(parameters)	Additional validation required of operation parameters not performed by JSON schema validator.

Attributes

RemoveColumnsOp.NAME

RemoveColumnsOp.PARAMS

RemoveColumnsOp.**__init__**(*parameters*)

Constructor for remove columns operation.

Parameters
parameters (*dict*) – Dictionary with the parameter values for required and optional parameters.

RemoveColumnsOp.**do_op**(*dispatcher*, *df*, *name*, *sidecar=None*)

Remove indicated columns from a dataframe.

Parameters

- **dispatcher** (*Dispatcher*) – Manages the operation I/O.
- **df** (*DataFrame*) – The DataFrame to be remodeled.
- **name** (*str*) – Unique identifier for the dataframe – often the original file path.
- **sidecar** (*Sidecar or file-like*) – Not needed for this operation.

Returns
A new dataframe after processing.

Return type
Dataframe

Raises
KeyError –

- If ignore_missing is False and a column not in the data is to be removed.

static RemoveColumnsOp.**validate_input_data**(*parameters*)

Additional validation required of operation parameters not performed by JSON schema validator.

RemoveColumnsOp.**NAME = 'remove_columns'**

RemoveColumnsOp.**PARAMS = {'additionalProperties': False, 'properties': {'column_names': {'items': {'type': 'string'}, 'minItems': 1, 'type': 'array', 'uniqueItems': True}, 'ignore_missing': {'type': 'boolean'}}, 'required': ['column_names', 'ignore_missing'], 'type': 'object'}**

3.4.3.4.12 remove_rows_op

Remove rows from a columnar file based on the values in a specified row.

Classes

RemoveRowsOp(parameters)	Remove rows from a columnar file based on the values in a specified row.

3.4.3.4.12.1 RemoveRowsOp

class RemoveRowsOp(*parameters*)

Remove rows from a columnar file based on the values in a specified row.

Required remodeling parameters:

- **column_name** (*str*): The name of column to be tested.
- **remove_values** (*list*): The values to test for row removal.

Methods

RemoveRowsOp.__init__(parameters)	Constructor for remove rows operation.
RemoveRowsOp.do_op(dispatcher, df, name[, ...])	Remove rows with the values indicated in the column.
RemoveRowsOp.validate_input_data(parameters)	Additional validation required of operation parameters not performed by JSON schema validator.

Attributes

RemoveRowsOp.NAME
RemoveRowsOp.PARAMS

RemoveRowsOp.**__init__**(*parameters*)

Constructor for remove rows operation.

Parameters
parameters (*dict*) – Dictionary with the parameter values for required and optional parameters.

RemoveRowsOp.**do_op**(*dispatcher*, *df*, *name*, *sidecar=None*)

Remove rows with the values indicated in the column.

Parameters

- **dispatcher** (*Dispatcher*) – Manages the operation I/O.
- **df** (*DataFrame*) – The DataFrame to be remodeled.
- **name** (*str*) – Unique identifier for the dataframe – often the original file path.
- **sidecar** (*Sidecar or file-like*) – Not needed for this operation.

Returns
A new dataframe after processing.

Return type
Dataframe

static RemoveRowsOp.**validate_input_data**(*parameters*)

Additional validation required of operation parameters not performed by JSON schema validator.

RemoveRowsOp.**NAME = 'remove_rows'**

RemoveRowsOp.**PARAMS = {'additionalProperties': False, 'properties': {'column_name': {'description': 'Name of the key column to determine which rows to remove.', 'type': 'string'}, 'remove_values': {'description': 'List of key values for rows to remove.', 'items': {'type': ['string', 'number']}, 'minItems': 1, 'type': 'array', 'uniqueItems': True}}, 'required': ['column_name', 'remove_values'], 'type': 'object'}**

3.4.3.4.13 rename_columns_op

Rename columns in a columnar file.

Classes

RenameColumnsOp(parameters)	Rename columns in a tabular file.

3.4.3.4.13.1 RenameColumnsOp

class RenameColumnsOp(*parameters*)

Rename columns in a tabular file.

Required remodeling parameters:

- **column_mapping** (*dict*): The names of the columns to be renamed with values to be remapped to.
- **ignore_missing** (*bool*): If true, the names in column_mapping that are not columns and should be ignored.

Methods

RenameColumnsOp.__init__(parameters)	Constructor for rename columns operation.
RenameColumnsOp.do_op(dispatcher, df, name)	Rename columns as specified in column_mapping dictionary.
RenameColumnsOp.validate_input_data(parameters)	Additional validation required of operation parameters not performed by JSON schema validator.

Attributes

RenameColumnsOp.NAME
RenameColumnsOp.PARAMS

RenameColumnsOp.**__init__**(*parameters*)

Constructor for rename columns operation.

Parameters
parameters (*dict*) – Dictionary with the parameter values for required and optional parameters

RenameColumnsOp.**do_op**(*dispatcher*, *df*, *name*, *sidecar=None*)

Rename columns as specified in column_mapping dictionary.

Parameters

- **dispatcher** (*Dispatcher*) – Manages the operation I/O.
- **df** (*DataFrame*) – The DataFrame to be remodeled.
- **name** (*str*) – Unique identifier for the dataframe – often the original file path.
- **sidecar** (*Sidecar or file-like*) – Not needed for this operation.

Returns
A new dataframe after processing.

Return type
Dataframe

Raises
KeyError –

- When ignore_missing is False and column_mapping has columns not in the data.

static RenameColumnsOp.**validate_input_data**(*parameters*)

Additional validation required of operation parameters not performed by JSON schema validator.

RenameColumnsOp.**NAME = 'rename_columns'**

RenameColumnsOp.**PARAMS = {'additionalProperties': False, 'properties': {'column_mapping': {'description': 'Mapping between original column names and their respective new names.', 'minProperties': 1, 'patternProperties': {'.*': {'type': 'string'}}, 'type': 'object'}, 'ignore_missing': {'description': "If true ignore column_mapping keys that don't correspond to columns, otherwise error.", 'type': 'boolean'}}, 'required': ['column_mapping', 'ignore_missing'], 'type': 'object'}**

3.4.3.4.14 reorder_columns_op

Reorder columns in a columnar file.

Classes

ReorderColumnsOp(parameters)	Reorder columns in a columnar file.

3.4.3.4.14.1 ReorderColumnsOp

class ReorderColumnsOp(*parameters*)

Reorder columns in a columnar file.

Required parameters:

- column_order (*list*): The names of the columns to be reordered.
- ignore_missing (*bool*): If False and a column in column_order is not in df, skip the column.
- keep_others (*bool*): If True, columns not in column_order are placed at end.

Methods

ReorderColumnsOp.__init__(parameters)	Constructor for reorder columns operation.
ReorderColumnsOp.do_op(dispatcher, df, name)	Reorder columns as specified in event dictionary.
ReorderColumnsOp.validate_input_data(parameters)	Additional validation required of operation parameters not performed by JSON schema validator.

Attributes

ReorderColumnsOp.NAME
ReorderColumnsOp.PARAMS

ReorderColumnsOp.**__init__**(*parameters*)

Constructor for reorder columns operation.

Parameters
parameters (*dict*) – Dictionary with the parameter values for required and optional parameters.

ReorderColumnsOp.**do_op**(*dispatcher*, *df*, *name*, *sidecar=None*)

Reorder columns as specified in event dictionary.

Parameters

- **dispatcher** (*Dispatcher*) – Manages the operation I/O.
- **df** (*DataFrame*) – The DataFrame to be remodeled.
- **name** (*str*) – Unique identifier for the dataframe – often the original file path.
- **sidecar** (*Sidecar or file-like*) – Not needed for this operation.

Returns
A new dataframe after processing.

Return type
Dataframe

Raises
ValueError –

- When ignore_missing is false and column_order has columns not in the data.

static ReorderColumnsOp.**validate_input_data**(*parameters*)

Additional validation required of operation parameters not performed by JSON schema validator.

ReorderColumnsOp.**NAME = 'reorder_columns'**

ReorderColumnsOp.**PARAMS = {'additionalProperties': False, 'properties': {'column_order': {'description': 'A list of column names in the order you wish them to be.', 'items': {'type': 'string'}, 'minItems': 1, 'type': 'array', 'uniqueItems': True}, 'ignore_missing': {'description': "If true, ignore column_order columns that aren't in file, otherwise error.", 'type': 'boolean'}, 'keep_others': {'description': 'If true columns not in column_order are placed at end, otherwise ignored.', 'type': 'boolean'}}, 'required': ['column_order', 'ignore_missing', 'keep_others'], 'type': 'object'}**

3.4.3.4.15 split_rows_op

Split rows in a columnar file with onset and duration columns into multiple rows based on a specified column.

Classes

SplitRowsOp(parameters)	Split rows in a columnar file with onset and duration columns into multiple rows based on a specified column.

3.4.3.4.15.1 SplitRowsOp

class SplitRowsOp(*parameters*)

Split rows in a columnar file with onset and duration columns into multiple rows based on a specified column.

Required remodeling parameters:

- **anchor_column** (*str*): The column in which the names of new items are stored.
- **new_events** (*dict*): Mapping of new values based on values in the original row.
- **remove_parent_row** (*bool*): If true, the original row that was split is removed.

Notes

- In specifying onset and duration for the new row, you can give values or the names of columns as strings.

Methods

SplitRowsOp.__init__(parameters)	Constructor for the split rows operation.
SplitRowsOp.do_op(dispatcher, df, name[, ...])	Split a row representing a particular event into multiple rows.
SplitRowsOp.validate_input_data(parameters)	Additional validation required of operation parameters not performed by JSON schema validator.

Attributes

SplitRowsOp.NAME

SplitRowsOp.PARAMS

SplitRowsOp.**__init__**(*parameters*)

Constructor for the split rows operation.

Parameters
parameters (*dict*) – Dictionary with the parameter values for required and optional parameters.

SplitRowsOp.**do_op**(*dispatcher*, *df*, *name*, *sidecar=None*)

Split a row representing a particular event into multiple rows.

Parameters

- **dispatcher** (*Dispatcher*) – Manages the operation I/O.
- **df** (*DataFrame*) – The DataFrame to be remodeled.
- **name** (*str*) – Unique identifier for the dataframe – often the original file path.
- **sidecar** (*Sidecar or file-like*) – Not needed for this operation.

Returns
A new dataframe after processing.

Return type
Dataframe

Raises
TypeError – -If bad onset or duration.

static SplitRowsOp.**validate_input_data**(*parameters*)

Additional validation required of operation parameters not performed by JSON schema validator.

SplitRowsOp.**NAME = 'split_rows'**

SplitRowsOp.**PARAMS = {'additionalProperties': False, 'properties': {'anchor_column': {'description': 'The column containing the keys for the new rows. (Original rows will have own keys.)', 'type': 'string'}, 'new_events': {'description': 'A map describing how the rows for the new codes will be created.', 'minProperties': 1, 'patternProperties': {'.*': {'additionalProperties': False, 'properties': {'copy_columns': {'description': 'List of columns whose values to copy for the new row.', 'items': {'type': 'string'}, 'minItems': 1, 'type': 'array', 'uniqueItems': True}, 'duration': {'description': 'List of items to add to compute the duration of the new row.', 'items': {'type': ['string', 'number']}, 'minItems': 1, 'type': 'array'}, 'onset_source': {'description': 'List of items to add to compute the onset time of the new row.', 'items': {'type': ['string', 'number']}, 'minItems': 1, 'type': 'array'}}, 'required': ['onset_source', 'duration'], 'type': 'object'}}, 'type': 'object'}, 'remove_parent_row': {'description': 'If true, the row from which these rows were split is removed, otherwise it stays.', 'type': 'boolean'}}, 'required': ['anchor_column', 'new_events', 'remove_parent_row'], 'type': 'object'}**

3.4.3.4.16 summarize_column_names_op

Summarize the column names in a collection of tabular files.

Classes

ColumnNamesSummary(sum_op)	Manager for summaries of column names for a dataset.
SummarizeColumnNamesOp(parameters)	Summarize the column names in a collection of tabular files.

3.4.3.4.16.1 ColumnNamesSummary

class ColumnNamesSummary(*sum_op*)

Manager for summaries of column names for a dataset.

Methods

ColumnNamesSummary.__init__(sum_op)	Constructor for column name summary manager.
ColumnNamesSummary.dump_summary(filename, ...)	
ColumnNamesSummary.get_details_dict(...)	Return the summary dictionary extracted from a ColumnNameSummary.
ColumnNamesSummary.get_individual(...[, ...])	Return a dictionary of the individual file summaries.
ColumnNamesSummary.get_summary([...])	Return a summary dictionary with the information.
ColumnNamesSummary.get_summary_details([...])	Return a dictionary with the details for individual files and the overall dataset.
ColumnNamesSummary.get_text_summary([...])	Return a complete text summary by assembling the individual pieces.
ColumnNamesSummary.get_text_summary_details([...])	Return a text summary of the information represented by this summary.
ColumnNamesSummary.merge_all_info()	Create a ColumnNameSummary containing the overall dataset summary.
ColumnNamesSummary.save(save_dir[, ...])	Save the summaries using the format indicated.
ColumnNamesSummary.save_visualizations(save_dir)	Save summary visualizations, if any, using the format indicated.
ColumnNamesSummary.update_summary(new_info)	Update the summary for a given tabular input file.

Attributes

ColumnNamesSummary.DISPLAY_INDENT
ColumnNamesSummary.INDIVIDUAL_SUMMARIES_PATH

ColumnNamesSummary.**__init__**(*sum_op*)

Constructor for column name summary manager.

Parameters
sum_op (*SummarizeColumnNamesOp*) – Operation associated with this summary.

static ColumnNamesSummary.**dump_summary**(*filename*, *summary*)

ColumnNamesSummary.**get_details_dict**(*column_summary*)

Return the summary dictionary extracted from a ColumnNameSummary.

Parameters
column_summary (*ColumnNameSummary*) – A column name summary for the data file.

Returns
dict - a dictionary with the summary information for column names.

ColumnNamesSummary.**get_individual**(*summary_details*, *separately=True*)

Return a dictionary of the individual file summaries.

Parameters

- **summary_details** (*dict*) – Dictionary of the individual file summaries.
- **separately** (*bool*) – If True (the default), each individual summary has a header for separate output.

ColumnNamesSummary.**get_summary**(*individual_summaries='separate'*)

Return a summary dictionary with the information.

Parameters
individual_summaries (*str*) – "separate", "consolidated", or "none"

Returns
dict - dictionary with "Dataset" and "Individual files" keys.

Notes: The individual_summaries value is processed as follows:

- "separate" individual summaries are to be in separate files.
- "consolidated" means that the individual summaries are in same file as overall summary.
- "none" means that only the overall summary is produced.

ColumnNamesSummary.**get_summary_details**(*include_individual=True*)

Return a dictionary with the details for individual files and the overall dataset.

Parameters
include_individual (*bool*) – If True, summaries for individual files are included.

Returns
dict - a dictionary with 'Dataset' and 'Individual files' keys.

Notes

- The 'Dataset' value is either a string or a dictionary with the overall summary.
- **The 'Individual files' value is dictionary whose keys are file names and values are** their corresponding summaries.

Users are expected to provide merge_all_info and get_details_dict functions to support this.

ColumnNamesSummary.**get_text_summary**(*individual_summaries='separate'*)

Return a complete text summary by assembling the individual pieces.

Parameters
individual_summaries (*str*) – One of the values "separate", "consolidated", or "none".

Returns
Complete text summary.

Return type
str

Notes: The options are:

- "none": Just has "Dataset" key.
- "consolidated" Has "Dataset" and "Individual files" keys with the values of each is a string.
- "separate" Has "Dataset" and "Individual files" keys. The values of "Individual files" is a dict.

ColumnNamesSummary.**get_text_summary_details**(*include_individual=True*)

Return a text summary of the information represented by this summary.

Parameters
include_individual (*bool*) – If True (the default), individual summaries are in "Individual files".

ColumnNamesSummary.**merge_all_info**()

Create a ColumnNameSummary containing the overall dataset summary.

Returns
ColumnNameSummary - the overall summary object for column names.

ColumnNamesSummary.**save**(*save_dir*, *file_formats=['.txt']*, *individual_summaries='separate'*, *task_name=''*)

Save the summaries using the format indicated.

Parameters

- **save_dir** (*str*) – Name of the directory to save the summaries in.
- **file_formats** (*list*) – List of file formats to use for saving.
- **individual_summaries** (*str*) – Save one file or multiple files based on setting.
- **task_name** (*str*) – If this summary corresponds to files from a task, the task_name is used in filename.

ColumnNamesSummary.**save_visualizations**(*save_dir*, *file_formats=['.svg']*, *individual_summaries='separate'*, *task_name=''*)

Save summary visualizations, if any, using the format indicated.

Parameters

- **save_dir** (*str*) – Name of the directory to save the summaries in.
- **file_formats** (*list*) – List of file formats to use for saving.
- **individual_summaries** (*str*) – Save one file or multiple files based on setting.
- **task_name** (*str*) – If this summary corresponds to files from a task, the task_name is used in filename.

ColumnNamesSummary.**update_summary**(*new_info*)

Update the summary for a given tabular input file.

Parameters
new_info (*dict*) – A dictionary with the parameters needed to update a summary.

Notes

- The summary information is kept in separate ColumnNameSummary objects for each file.
- The summary needs a "name" str and a "column_names" list.
- The summary uses ColumnNameSummary as the summary object.

ColumnNamesSummary.**DISPLAY_INDENT** = ' '

ColumnNamesSummary.**INDIVIDUAL_SUMMARIES_PATH** = 'individual_summaries'

3.4.3.4.16.2 SummarizeColumnNamesOp

class SummarizeColumnNamesOp(*parameters*)

Summarize the column names in a collection of tabular files.

Required remodeling parameters:

- **summary_name** (*str*): The name of the summary.
- **summary_filename** (*str*): Base filename of the summary.

Optional remodeling parameters:

- **append_timecode** (*bool*): If False (default), the timecode is not appended to the summary filename.

The purpose is to check that all the tabular files have the same columns in same order.

Methods

SummarizeColumnNamesOp.__init__(parameters)	Constructor for summarize column names operation.
SummarizeColumnNamesOp.do_op(dispatcher, df, ...)	Create a column name summary for df.
SummarizeColumnNamesOp.validate_input_data(...)	Additional validation required of operation parameters not performed by JSON schema validator.

Attributes

SummarizeColumnNamesOp.NAME
SummarizeColumnNamesOp.PARAMS
SummarizeColumnNamesOp.SUMMARY_TYPE

SummarizeColumnNamesOp.**__init__**(*parameters*)

Constructor for summarize column names operation.

Parameters
parameters (*dict*) – Dictionary with the parameter values for required and optional parameters.

SummarizeColumnNamesOp.**do_op**(*dispatcher*, *df*, *name*, *sidecar=None*)

Create a column name summary for df.

Parameters

- **dispatcher** (*Dispatcher*) – Manages the operation I/O.
- **df** (*DataFrame*) – The DataFrame to be remodeled.
- **name** (*str*) – Unique identifier for the dataframe – often the original file path.
- **sidecar** (*Sidecar or file-like*) – Not needed for this operation.

Returns
A copy of df.

Return type
DataFrame

Side effect:
Updates the relevant summary.

static SummarizeColumnNamesOp.**validate_input_data**(*parameters*)

Additional validation required of operation parameters not performed by JSON schema validator.

SummarizeColumnNamesOp.**NAME = 'summarize_column_names'**

SummarizeColumnNamesOp.**PARAMS = {'additionalProperties': False, 'properties': {'append_timecode': {'description': 'If true, the timecode is appended to the base filename so each run has a unique name.', 'type': 'boolean'}, 'summary_filename': {'description': 'Name to use for the summary file name base.', 'type': 'string'}, 'summary_name': {'description': 'Name to use for the summary in titles.', 'type': 'string'}}, 'required': ['summary_name', 'summary_filename'], 'type': 'object'}**

SummarizeColumnNamesOp.**SUMMARY_TYPE = 'column_names'**

3.4.3.4.17 summarize_column_values_op

Summarize the values in the columns of a columnar file.

Classes

ColumnValueSummary(sum_op)	Manager for summaries of column contents for columnar files.
SummarizeColumnValuesOp(parameters)	Summarize the values in the columns of a columnar file.

3.4.3.4.17.1 ColumnValueSummary

class ColumnValueSummary(*sum_op*)

Manager for summaries of column contents for columnar files.

Methods

ColumnValueSummary.__init__(sum_op)	Constructor for column value summary manager.
ColumnValueSummary.dump_summary(filename, ...)	
ColumnValueSummary.get_details_dict(summary)	Return a dictionary with the summary contained in a TabularSummary.
ColumnValueSummary.get_individual(...[, ...])	Return a dictionary of the individual file summaries.
ColumnValueSummary.get_list_str(lst)	Return a str version of a list with items separated by a blank.
ColumnValueSummary.get_summary([...])	Return a summary dictionary with the information.
ColumnValueSummary.get_summary_details([...])	Return a dictionary with the details for individual files and the overall dataset.
ColumnValueSummary.get_text_summary([...])	Return a complete text summary by assembling the individual pieces.
ColumnValueSummary.get_text_summary_details([...])	Return a text summary of the information represented by this summary.
ColumnValueSummary.merge_all_info()	Create a TabularSummary containing the overall dataset summary.
ColumnValueSummary.partition_list(lst, n)	Partition a list into lists of n items.
ColumnValueSummary.save(save_dir[, ...])	Save the summaries using the format indicated.
ColumnValueSummary.save_visualizations(save_dir)	Save summary visualizations, if any, using the format indicated.
ColumnValueSummary.sort_dict(count_dict[, ...])	
ColumnValueSummary.update_summary(new_info)	Update the summary for a given tabular input file.

Attributes

ColumnValueSummary.DISPLAY_INDENT
ColumnValueSummary.INDIVIDUAL_SUMMARIES_PATH

ColumnValueSummary.**__init__**(*sum_op*)

Constructor for column value summary manager.

Parameters

sum_op (*SummarizeColumnValuesOp*) – Operation associated with this summary.

static ColumnValueSummary.**dump_summary**(*filename*, *summary*)

ColumnValueSummary.**get_details_dict**(*summary*)

Return a dictionary with the summary contained in a TabularSummary.

Parameters
summary (*TabularSummary*) – Dictionary of merged summary information.

Returns
Dictionary with the information suitable for extracting printout.

Return type
dict

ColumnValueSummary.**get_individual**(*summary_details*, *separately=True*)

Return a dictionary of the individual file summaries.

Parameters

- **summary_details** (*dict*) – Dictionary of the individual file summaries.
- **separately** (*bool*) – If True (the default), each individual summary has a header for separate output.

static ColumnValueSummary.**get_list_str**(*lst*)

Return a str version of a list with items separated by a blank.

Returns
String version of list.

Return type
str

ColumnValueSummary.**get_summary**(*individual_summaries='separate'*)

Return a summary dictionary with the information.

Parameters
individual_summaries (*str*) – "separate", "consolidated", or "none"

Returns
dict - dictionary with "Dataset" and "Individual files" keys.

Notes: The individual_summaries value is processed as follows:

- "separate" individual summaries are to be in separate files.
- "consolidated" means that the individual summaries are in same file as overall summary.
- "none" means that only the overall summary is produced.

ColumnValueSummary.**get_summary_details**(*include_individual=True*)

Return a dictionary with the details for individual files and the overall dataset.

Parameters
include_individual (*bool*) – If True, summaries for individual files are included.

Returns
dict - a dictionary with 'Dataset' and 'Individual files' keys.

Notes

- The 'Dataset' value is either a string or a dictionary with the overall summary.
- **The 'Individual files' value is dictionary whose keys are file names and values are** their corresponding summaries.

Users are expected to provide merge_all_info and get_details_dict functions to support this.

ColumnValueSummary.**get_text_summary**(*individual_summaries='separate'*)

Return a complete text summary by assembling the individual pieces.

Parameters
individual_summaries (*str*) – One of the values "separate", "consolidated", or "none".

Returns
Complete text summary.

Return type
str

Notes: The options are:

- "none": Just has "Dataset" key.
- "consolidated" Has "Dataset" and "Individual files" keys with the values of each is a string.
- "separate" Has "Dataset" and "Individual files" keys. The values of "Individual files" is a dict.

ColumnValueSummary.**get_text_summary_details**(*include_individual=True*)

Return a text summary of the information represented by this summary.

Parameters
include_individual (*bool*) – If True (the default), individual summaries are in "Individual files".

ColumnValueSummary.**merge_all_info**()

Create a TabularSummary containing the overall dataset summary.

Returns
TabularSummary - the summary object for column values.

static ColumnValueSummary.**partition_list**(*lst*, *n*)

Partition a list into lists of n items.

Parameters

- **lst** (*list*) – List to be partitioned.
- **n** (*int*) – Number of items in each sublist.

Returns
list of lists of n elements, the last might have fewer.

Return type
list

ColumnValueSummary.**save**(*save_dir*, *file_formats=['.txt']*, *individual_summaries='separate'*, *task_name=''*)

Save the summaries using the format indicated.

Parameters

- **save_dir** (*str*) – Name of the directory to save the summaries in.

- **file_formats** (*list*) – List of file formats to use for saving.
- **individual_summaries** (*str*) – Save one file or multiple files based on setting.
- **task_name** (*str*) – If this summary corresponds to files from a task, the task_name is used in filename.

ColumnValueSummary.**save_visualizations**(*save_dir*, *file_formats=['.svg']*, *individual_summaries='separate'*, *task_name=''*)

Save summary visualizations, if any, using the format indicated.

Parameters

- **save_dir** (*str*) – Name of the directory to save the summaries in.
- **file_formats** (*list*) – List of file formats to use for saving.
- **individual_summaries** (*str*) – Save one file or multiple files based on setting.
- **task_name** (*str*) – If this summary corresponds to files from a task, the task_name is used in filename.

static ColumnValueSummary.**sort_dict**(*count_dict*, *reverse=False*)

ColumnValueSummary.**update_summary**(*new_info*)

Update the summary for a given tabular input file.

Parameters
new_info (*dict*) – A dictionary with the parameters needed to update a summary.

Notes

- The summary information is kept in separate TabularSummary objects for each file.
- The summary needs a "name" str and a "df" .

ColumnValueSummary.**DISPLAY_INDENT = ' '**

ColumnValueSummary.**INDIVIDUAL_SUMMARIES_PATH = 'individual_summaries'**

3.4.3.4.17.2 SummarizeColumnValuesOp

class SummarizeColumnValuesOp(*parameters*)

Summarize the values in the columns of a columnar file.

Required remodeling parameters:

- **summary_name** (*str*): The name of the summary.
- **summary_filename** (*str*): Base filename of the summary.

Optional remodeling parameters:

- **append_timecode** (*bool*): (**Optional**: Default False) If True append timecodes to the summary filename.
- **max_categorical** (*int*): Maximum number of unique values to include in summary for a categorical column.
- **skip_columns** (*list*): Names of columns to skip in the summary.

- **value_columns** (*list*): Names of columns to treat as value columns rather than categorical columns.
- **values_per_line** (*int*): The number of values output per line in the summary.

The purpose is to produce a summary of the values in a tabular file.

Methods

SummarizeColumnValuesOp.__init__(parameters)	Constructor for the summarize column values operation.
SummarizeColumnValuesOp.do_op(dispatcher, ...)	Create a summary of the column values in df.
SummarizeColumnValuesOp.validate_input_data(...)	Additional validation required of operation parameters not performed by JSON schema validator.

Attributes

SummarizeColumnValuesOp.MAX_CATEGORICAL
SummarizeColumnValuesOp.NAME
SummarizeColumnValuesOp.PARAMS
SummarizeColumnValuesOp.SUMMARY_TYPE
SummarizeColumnValuesOp.VALUES_PER_LINE

SummarizeColumnValuesOp.**__init__**(*parameters*)

Constructor for the summarize column values operation.

Parameters
parameters (*dict*) – Dictionary with the parameter values for required and optional parameters.

SummarizeColumnValuesOp.**do_op**(*dispatcher*, *df*, *name*, *sidecar=None*)

Create a summary of the column values in df.

Parameters

- **dispatcher** (*Dispatcher*) – Manages the operation I/O.
- **df** (*DataFrame*) – The DataFrame to be remodeled.
- **name** (*str*) – Unique identifier for the dataframe – often the original file path.
- **sidecar** (*Sidecar or file-like*) – Not needed for this operation.

Returns
A copy of df.

Return type
DataFrame

Side effect:
Updates the relevant summary.

static SummarizeColumnValuesOp.**validate_input_data**(*parameters*)

Additional validation required of operation parameters not performed by JSON schema validator.

SummarizeColumnValuesOp.**MAX_CATEGORICAL = 50**

SummarizeColumnValuesOp.**NAME = 'summarize_column_values'**

SummarizeColumnValuesOp.**PARAMS = {'additionalProperties': False, 'properties': {'append_timecode': {'description': 'If true, the timecode is appended to the base filename so each run has a unique name.', 'type': 'boolean'}, 'max_categorical': {'description': 'Maximum number of unique column values to show in text description.', 'type': 'integer'}, 'skip_columns': {'description': 'List of columns to skip when creating the summary.', 'items': {'type': 'string'}, 'minItems': 1, 'type': 'array', 'uniqueItems': True}, 'summary_filename': {'description': 'Name to use for the summary file name base.', 'type': 'string'}, 'summary_name': {'description': 'Name to use for the summary in titles.', 'type': 'string'}, 'value_columns': {'description': 'Columns to be annotated with a single HED annotation and placeholder.', 'items': {'type': 'string'}, 'minItems': 1, 'type': 'array', 'uniqueItems': True}, 'values_per_line': {'description': 'Number of items per line to display in the text file.', 'type': 'integer'}}, 'required': ['summary_name', 'summary_filename'], 'type': 'object'}**

SummarizeColumnValuesOp.**SUMMARY_TYPE = 'column_values'**

SummarizeColumnValuesOp.**VALUES_PER_LINE = 5**

3.4.3.4.18 summarize_definitions_op

Summarize the type_defs in the dataset.

Classes

DefinitionSummary(sum_op, hed_schema[, ...])	Manager for summaries of the definitions used in a dataset.
SummarizeDefinitionsOp(parameters)	Summarize the definitions used in the dataset based on Def and Def-expand.

3.4.3.4.18.1 DefinitionSummary

class DefinitionSummary(*sum_op*, *hed_schema*, *known_defs=None*)

Manager for summaries of the definitions used in a dataset.

Methods

DefinitionSummary.__init__(sum_op, hed_schema)	Constructor for the summary of definitions.
DefinitionSummary.dump_summary(filename, summary)	
DefinitionSummary.get_details_dict(def_gatherer)	Return the summary-specific information in a dictionary.
DefinitionSummary.get_individual(summary_details)	Return a dictionary of the individual file summaries.
DefinitionSummary.get_summary([...])	Return a summary dictionary with the information.
DefinitionSummary.get_summary_details([...])	Return a dictionary with the details for individual files and the overall dataset.
DefinitionSummary.get_text_summary([...])	Return a complete text summary by assembling the individual pieces.
DefinitionSummary.get_text_summary_details([...])	Return a text summary of the information represented by this summary.
DefinitionSummary.merge_all_info()	Create an Object containing the definition summary.
DefinitionSummary.save(save_dir[, ...])	Save the summaries using the format indicated.
DefinitionSummary.save_visualizations(save_dir)	Save summary visualizations, if any, using the format indicated.
DefinitionSummary.update_summary(new_info)	Update the summary for a given tabular input file.

Attributes

DefinitionSummary.DISPLAY_INDENT
DefinitionSummary.INDIVIDUAL_SUMMARIES_PATH

DefinitionSummary.**__init__**(*sum_op*, *hed_schema*, *known_defs=None*)

Constructor for the summary of definitions.

Parameters

- **sum_op** (*SummarizeDefinitionsOp*) – Summary operation class for gathering definitions.
- **hed_schema** (*HedSchema or HedSchemaGroup*) – Schema used for the dataset.
- **known_defs** (*str or list or DefinitionDict*) – Definitions already known to be used.

static DefinitionSummary.**dump_summary**(*filename*, *summary*)

DefinitionSummary.**get_details_dict**(*def_gatherer*)

Return the summary-specific information in a dictionary.

Parameters

def_gatherer (*DefExpandGatherer*) – Contains the resolved dictionaries.

Returns

dictionary with the summary results.

Return type

dict

DefinitionSummary.**get_individual**(*summary_details*, *separately=True*)

Return a dictionary of the individual file summaries.

Parameters

- **summary_details** (*dict*) – Dictionary of the individual file summaries.
- **separately** (*bool*) – If True (the default), each individual summary has a header for separate output.

DefinitionSummary.**get_summary**(*individual_summaries='separate'*)

Return a summary dictionary with the information.

Parameters
individual_summaries (*str*) – "separate", "consolidated", or "none"

Returns
dict - dictionary with "Dataset" and "Individual files" keys.

Notes: The individual_summaries value is processed as follows:

- "separate" individual summaries are to be in separate files.
- "consolidated" means that the individual summaries are in same file as overall summary.
- "none" means that only the overall summary is produced.

DefinitionSummary.**get_summary_details**(*include_individual=True*)

Return a dictionary with the details for individual files and the overall dataset.

Parameters
include_individual (*bool*) – If True, summaries for individual files are included.

Returns
dict - a dictionary with 'Dataset' and 'Individual files' keys.

Notes

- The 'Dataset' value is either a string or a dictionary with the overall summary.
- **The 'Individual files' value is dictionary whose keys are file names and values are** their corresponding summaries.

Users are expected to provide merge_all_info and get_details_dict functions to support this.

DefinitionSummary.**get_text_summary**(*individual_summaries='separate'*)

Return a complete text summary by assembling the individual pieces.

Parameters
individual_summaries (*str*) – One of the values "separate", "consolidated", or "none".

Returns
Complete text summary.

Return type
str

Notes: The options are:

- "none": Just has "Dataset" key.

- "consolidated" Has "Dataset" and "Individual files" keys with the values of each is a string.
- "separate" Has "Dataset" and "Individual files" keys. The values of "Individual files" is a dict.

DefinitionSummary.**get_text_summary_details**(*include_individual=True*)

Return a text summary of the information represented by this summary.

Parameters

include_individual (*bool*) – If True (the default), individual summaries are in "Individual files".

DefinitionSummary.**merge_all_info**()

Create an Object containing the definition summary.

Returns

Object - the overall summary object for type_defs.

DefinitionSummary.**save**(*save_dir*, *file_formats=['.txt']*, *individual_summaries='separate'*, *task_name=''*)

Save the summaries using the format indicated.

Parameters

- **save_dir** (*str*) – Name of the directory to save the summaries in.
- **file_formats** (*list*) – List of file formats to use for saving.
- **individual_summaries** (*str*) – Save one file or multiple files based on setting.
- **task_name** (*str*) – If this summary corresponds to files from a task, the task_name is used in filename.

DefinitionSummary.**save_visualizations**(*save_dir*, *file_formats=['.svg']*, *individual_summaries='separate'*, *task_name=''*)

Save summary visualizations, if any, using the format indicated.

Parameters

- **save_dir** (*str*) – Name of the directory to save the summaries in.
- **file_formats** (*list*) – List of file formats to use for saving.
- **individual_summaries** (*str*) – Save one file or multiple files based on setting.
- **task_name** (*str*) – If this summary corresponds to files from a task, the task_name is used in filename.

DefinitionSummary.**update_summary**(*new_info*)

Update the summary for a given tabular input file.

Parameters

new_info (*dict*) – A dictionary with the parameters needed to update a summary.

Notes

- The summary needs a "name" str, a "schema" and a "Sidecar".

DefinitionSummary.**DISPLAY_INDENT = ' '**

DefinitionSummary.**INDIVIDUAL_SUMMARIES_PATH = 'individual_summaries'**

3.4.3.4.18.2 SummarizeDefinitionsOp

class SummarizeDefinitionsOp(*parameters*)

Summarize the definitions used in the dataset based on Def and Def-expand.

Required remodeling parameters:

- **summary_name** (*str*): The name of the summary.
- **summary_filename** (*str*): Base filename of the summary.

Optional remodeling parameters:

- **append_timecode** (*bool*): If False (default), the timecode is not appended to the summary filename.

The purpose is to produce a summary of the definitions used in a dataset.

Methods

SummarizeDefinitionsOp.__init__(parameters)	Constructor for the summary of definitions used in the dataset.
SummarizeDefinitionsOp.do_op(dispatcher, df, ...)	Create summaries of definitions.
SummarizeDefinitionsOp.validate_input_data(...)	Additional validation required of operation parameters not performed by JSON schema validator.

Attributes

SummarizeDefinitionsOp.NAME
SummarizeDefinitionsOp.PARAMS
SummarizeDefinitionsOp.SUMMARY_TYPE

SummarizeDefinitionsOp.**__init__**(*parameters*)

Constructor for the summary of definitions used in the dataset.

Parameters
parameters (*dict*) – Dictionary with the parameter values for required and optional parameters.

SummarizeDefinitionsOp.**do_op**(*dispatcher*, *df*, *name*, *sidecar=None*)

Create summaries of definitions.

Parameters

- **dispatcher** (*Dispatcher*) – Manages the operation I/O.

- **df** (*DataFrame*) – The DataFrame to be remodeled.
- **name** (*str*) – Unique identifier for the dataframe – often the original file path.
- **sidecar** (*Sidecar or file-like*) – Only needed for HED operations.

Returns
a copy of df

Return type
DataFrame

Side effect:
Updates the relevant summary.

static SummarizeDefinitionsOp.**validate_input_data**(*parameters*)

Additional validation required of operation parameters not performed by JSON schema validator.

SummarizeDefinitionsOp.**NAME = 'summarize_definitions'**

SummarizeDefinitionsOp.**PARAMS = {'additionalProperties': False, 'properties': {'append_timecode': {'description': 'If true, the timecode is appended to the base filename so each run has a unique name.', 'type': 'boolean'}, 'summary_filename': {'description': 'Name to use for the summary file name base.', 'type': 'string'}, 'summary_name': {'description': 'Name to use for the summary in titles.', 'type': 'string'}}, 'required': ['summary_name', 'summary_filename'], 'type': 'object'}**

SummarizeDefinitionsOp.**SUMMARY_TYPE = 'type_defs'**

3.4.3.4.19 summarize_hed_tags_op

Summarize the HED tags in collection of tabular files.

Classes

HedTagSummary(sum_op)	Manager of the HED tag summaries.
SummarizeHedTagsOp(parameters)	Summarize the HED tags in collection of tabular files.

3.4.3.4.19.1 HedTagSummary

class HedTagSummary(*sum_op*)

Manager of the HED tag summaries.

Methods

HedTagSummary.__init__(sum_op)	Constructor for HED tag summary manager.
HedTagSummary.dump_summary(filename, summary)	
HedTagSummary.get_details_dict(tag_counts)	Return the summary-specific information in a dictionary.
HedTagSummary.get_individual(summary_details)	Return a dictionary of the individual file summaries.
HedTagSummary.get_summary([individual_summaries])	Return a summary dictionary with the information.
HedTagSummary.get_summary_details([...])	Return a dictionary with the details for individual files and the overall dataset.
HedTagSummary.get_text_summary([...])	Return a complete text summary by assembling the individual pieces.
HedTagSummary.get_text_summary_details([...])	Return a text summary of the information represented by this summary.
HedTagSummary.merge_all_info()	Create a HedTagCounts containing the overall dataset HED tag summary.
HedTagSummary.save(save_dir[, file_formats, ...])	Save the summaries using the format indicated.
HedTagSummary.save_visualizations(save_dir)	Save the summary visualizations if any.
HedTagSummary.summary_to_dict(specifics[, ...])	Convert a HedTagSummary json specifics dict into the word cloud input format.
HedTagSummary.update_summary(new_info)	Update the summary for a given tabular input file.

Attributes

HedTagSummary.DISPLAY_INDENT
HedTagSummary.INDIVIDUAL_SUMMARIES_PATH

HedTagSummary.**__init__**(*sum_op*)

Constructor for HED tag summary manager.

Parameters
sum_op (*SummarizeHedTagsOp*) – Operation associated with this summary.

static HedTagSummary.**dump_summary**(*filename*, *summary*)

HedTagSummary.**get_details_dict**(*tag_counts*)

Return the summary-specific information in a dictionary.

Parameters
tag_counts (*HedTagCounts*) – Contains the counts of tags in the dataset.

Returns
dictionary with the summary results.

Return type
dict

HedTagSummary.**get_individual**(*summary_details*, *separately=True*)

Return a dictionary of the individual file summaries.

Parameters

- **summary_details** (*dict*) – Dictionary of the individual file summaries.
- **separately** (*bool*) – If True (the default), each individual summary has a header for separate output.

HedTagSummary.**get_summary**(*individual_summaries='separate'*)

Return a summary dictionary with the information.

Parameters
individual_summaries (*str*) – "separate", "consolidated", or "none"

Returns
dict - dictionary with "Dataset" and "Individual files" keys.

Notes: The individual_summaries value is processed as follows:

- "separate" individual summaries are to be in separate files.
- "consolidated" means that the individual summaries are in same file as overall summary.
- "none" means that only the overall summary is produced.

HedTagSummary.**get_summary_details**(*include_individual=True*)

Return a dictionary with the details for individual files and the overall dataset.

Parameters
include_individual (*bool*) – If True, summaries for individual files are included.

Returns
dict - a dictionary with 'Dataset' and 'Individual files' keys.

Notes

- The 'Dataset' value is either a string or a dictionary with the overall summary.
- **The 'Individual files' value is dictionary whose keys are file names and values are** their corresponding summaries.

Users are expected to provide merge_all_info and get_details_dict functions to support this.

HedTagSummary.**get_text_summary**(*individual_summaries='separate'*)

Return a complete text summary by assembling the individual pieces.

Parameters
individual_summaries (*str*) – One of the values "separate", "consolidated", or "none".

Returns
Complete text summary.

Return type
str

Notes: The options are:

- "none": Just has "Dataset" key.
- "consolidated" Has "Dataset" and "Individual files" keys with the values of each is a string.
- "separate" Has "Dataset" and "Individual files" keys. The values of "Individual files" is a dict.

HedTagSummary.**get_text_summary_details**(*include_individual=True*)

Return a text summary of the information represented by this summary.

Parameters

include_individual (*bool*) – If True (the default), individual summaries are in "Individual files".

HedTagSummary.**merge_all_info**()

Create a HedTagCounts containing the overall dataset HED tag summary.

Returns

The overall dataset summary object for HED tag counts.

Return type

HedTagCounts

HedTagSummary.**save**(*save_dir*, *file_formats=['.txt']*, *individual_summaries='separate'*, *task_name=''*)

Save the summaries using the format indicated.

Parameters

- **save_dir** (*str*) – Name of the directory to save the summaries in.
- **file_formats** (*list*) – List of file formats to use for saving.
- **individual_summaries** (*str*) – Save one file or multiple files based on setting.
- **task_name** (*str*) – If this summary corresponds to files from a task, the task_name is used in filename.

HedTagSummary.**save_visualizations**(*save_dir*, *file_formats=['.svg']*, *individual_summaries='separate'*, *task_name=''*)

Save the summary visualizations if any.

Parameters

- **save_dir** (*str*) – Path to directory in which visualizations should be saved.
- **file_formats** (*list*) – List of file formats to use in saving.
- **individual_summaries** (*str*) – One of "consolidated", "separate", or "none" indicating what to save.
- **task_name** (*str*) – Name of task if segregated by task.

static HedTagSummary.**summary_to_dict**(*specifics*, *transform=<ufunc 'log10'>*, *scale_adjustment=7*)

Convert a HedTagSummary json specifics dict into the word cloud input format.

Parameters

- **specifics** (*dict*) – Dictionary with keys "Main tags" and "Other tags".
- **transform** (*func*) – The function to transform the number of found tags. Default log10
- **scale_adjustment** (*int*) – Value added after transform.

Returns

a dict of the words and their occurrence count.

Return type

word_dict(dict)

Raises

KeyError – A malformed dictionary was passed.

HedTagSummary.**update_summary**(*new_info*)

Update the summary for a given tabular input file.

Parameters
new_info (*dict*) – A dictionary with the parameters needed to update a summary.

Notes

- The summary needs a "name" str, a "schema", a "df, and a "Sidecar".

HedTagSummary.**DISPLAY_INDENT = ' '**

HedTagSummary.**INDIVIDUAL_SUMMARIES_PATH = 'individual_summaries'**

3.4.3.4.19.2 SummarizeHedTagsOp

class SummarizeHedTagsOp(*parameters*)

Summarize the HED tags in collection of tabular files.

Required remodeling parameters:

- **summary_name** (*str*): The name of the summary.
- **summary_filename** (*str*): Base filename of the summary.
- **tags** (*dict*): Specifies how to organize the tag output.

Optional remodeling parameters:

- **append_timecode** (*bool*): If True, the timecode is appended to the base filename when summary is saved.
- **include_context** (*bool*): If True, context of events is included in summary.
- **remove_types** (*list*): A list of type tags such as Condition-variable or Task to exclude from summary.
- **replace_defs** (*bool*): If True, the def tag is replaced by the contents of the definitions.
- **word_cloud** (*bool*): If True, output a word cloud visualization.

The purpose of this op is to produce a summary of the occurrences of HED tags organized in a specified manner.

Notes: The tags template is a dictionary whose keys are the organization titles (not necessarily tags) for the output and whose values are the tags, which if they or their children appear, they will be listed under that title.

Methods

SummarizeHedTagsOp.__init__(parameters)	Constructor for the summarize_hed_tags operation.
SummarizeHedTagsOp.do_op(dispatcher, df, name)	Summarize the HED tags present in the dataset.
SummarizeHedTagsOp.validate_input_data(...)	Additional validation required of operation parameters not performed by JSON schema validator.

Attributes

SummarizeHedTagsOp.NAME

SummarizeHedTagsOp.PARAMS

SummarizeHedTagsOp.SUMMARY_TYPE

SummarizeHedTagsOp.**__init__**(*parameters*)

Constructor for the summarize_hed_tags operation.

Parameters
parameters (*dict*) – Dictionary with the parameter values for required and optional parameters.

SummarizeHedTagsOp.**do_op**(*dispatcher*, *df*, *name*, *sidecar=None*)

Summarize the HED tags present in the dataset.

Parameters

- **dispatcher** (*Dispatcher*) – Manages the operation I/O.
- **df** (*DataFrame*) – The DataFrame to be remodeled.
- **name** (*str*) – Unique identifier for the dataframe – often the original file path.
- **sidecar** (*Sidecar or file-like*) – Only needed for HED operations.

Returns
A copy of df.

Return type
DataFrame

Side effect:
Updates the context.

static SummarizeHedTagsOp.**validate_input_data**(*parameters*)

Additional validation required of operation parameters not performed by JSON schema validator.

SummarizeHedTagsOp.**NAME = 'summarize_hed_tags'**

SummarizeHedTagsOp.**PARAMS** = {'additionalProperties': False, 'properties': {'append_timecode': {'description': 'If true, the timecode is appended to the base filename so each run has a unique name.', 'type': 'boolean'}, 'include_context': {'description': 'If true, tags for events that unfold over time are counted at each intermediate time.', 'type': 'boolean'}, 'remove_types': {'description': 'A list of special tags such as Condition-variable whose influence is to be removed.', 'items': {'type': 'string'}, 'minItems': 1, 'type': 'array', 'uniqueItems': True}, 'replace_defs': {'description': 'If true, then the Def tags are replaced with actual definitions for the count.', 'type': 'boolean'}, 'summary_filename': {'description': 'Name to use for the summary file name base.', 'type': 'string'}, 'summary_name': {'description': 'Name to use for the summary in titles.', 'type': 'string'}, 'tags': {'description': 'A dictionary with the template for how output of tags should be organized.', 'patternProperties': {'.*': {'items': {'type': 'string'}, 'minItems': 1, 'type': 'array', 'uniqueItems': True}, 'additionalProperties': False, 'minProperties': 1}, 'type': 'object'}, 'word_cloud': {'additionalProperties': False, 'properties': {'background_color': {'description': 'Name of the background color (uses MatPlotLib names for colors).', 'type': 'string'}, 'contour_color': {'description': 'Name of the contour color (uses MatPlotLib names for colors).', 'type': 'string'}, 'contour_width': {'description': 'Width in pixels of contour surrounding the words.', 'type': 'number'}, 'font_path': {'description': 'Path to system font to use for word cloud display (system-specific).', 'type': 'string'}, 'height': {'description': 'Height of word cloud image in pixels.', 'type': 'integer'}, 'mask_path': {'description': 'Path of the mask image used to surround the words.', 'type': 'string'}, 'max_font_size': {'description': 'Maximum font size in point for the word cloud words.', 'type': 'number'}, 'min_font_size': {'description': 'Minimum font size in points for the word cloud words.', 'type': 'number'}, 'prefer_horizontal': {'description': 'Fraction of the words that are oriented horizontally.', 'type': 'number'}, 'scale_adjustment': {'description': 'Constant to add to log-transformed frequencies of the words to get scale.', 'type': 'number'}, 'set_font': {'description': 'If true, set the font to a system font (provided by font_path).', 'type': 'boolean'}, 'use_mask': {'description': 'If true then confine the word display to region within the provided mask.', 'type': 'boolean'}, 'width': {'description': 'Width of word cloud image in pixels.', 'type': 'integer'}}, 'type': 'object'}}, 'required': ['summary_name', 'summary_filename', 'tags'], 'type': 'object'}

SummarizeHedTagsOp.**SUMMARY_TYPE** = 'hed_tag_summary'

3.4.3.4.20 summarize_hed_type_op

Summarize a HED type tag in a collection of tabular files.

Classes

HedTypeSummary(sum_op)	Manager of the HED type summaries.
SummarizeHedTypeOp(parameters)	Summarize a HED type tag in a collection of tabular files.

3.4.3.4.20.1 HedTypeSummary

class HedTypeSummary(*sum_op*)

Manager of the HED type summaries.

Methods

HedTypeSummary.__init__(sum_op)	Constructor for HED type summary manager.
HedTypeSummary.dump_summary(filename, summary)	
HedTypeSummary.get_details_dict(counts)	Return the summary-specific information in a dictionary.
HedTypeSummary.get_individual(summary_details)	Return a dictionary of the individual file summaries.
HedTypeSummary.get_summary([...])	Return a summary dictionary with the information.
HedTypeSummary.get_summary_details([...])	Return a dictionary with the details for individual files and the overall dataset.
HedTypeSummary.get_text_summary([...])	Return a complete text summary by assembling the individual pieces.
HedTypeSummary.get_text_summary_details([...])	Return a text summary of the information represented by this summary.
HedTypeSummary.merge_all_info()	Create a HedTypeCounts containing the overall dataset HED type summary.
HedTypeSummary.save(save_dir[, ...])	Save the summaries using the format indicated.
HedTypeSummary.save_visualizations(save_dir)	Save summary visualizations, if any, using the format indicated.
HedTypeSummary.update_summary(new_info)	Update the summary for a given tabular input file.

Attributes

HedTypeSummary.DISPLAY_INDENT
HedTypeSummary.INDIVIDUAL_SUMMARIES_PATH

HedTypeSummary.**__init__**(*sum_op*)

Constructor for HED type summary manager.

Parameters
sum_op (*SummarizeHedTypeOp*) – Operation associated with this summary.

static HedTypeSummary.**dump_summary**(*filename*, *summary*)

HedTypeSummary.**get_details_dict**(*counts*)

Return the summary-specific information in a dictionary.

Parameters
counts (*HedTypeCounts*) – Contains the counts of the events in which the type occurs.

Returns
dictionary with the summary results.

Return type
dict

HedTypeSummary.**get_individual**(*summary_details*, *separately=True*)

Return a dictionary of the individual file summaries.

Parameters

- **summary_details** (*dict*) – Dictionary of the individual file summaries.
- **separately** (*bool*) – If True (the default), each individual summary has a header for separate output.

HedTypeSummary.**get_summary**(*individual_summaries='separate'*)

Return a summary dictionary with the information.

Parameters
individual_summaries (*str*) – "separate", "consolidated", or "none"

Returns
dict - dictionary with "Dataset" and "Individual files" keys.

Notes: The individual_summaries value is processed as follows:

- "separate" individual summaries are to be in separate files.
- "consolidated" means that the individual summaries are in same file as overall summary.
- "none" means that only the overall summary is produced.

HedTypeSummary.**get_summary_details**(*include_individual=True*)

Return a dictionary with the details for individual files and the overall dataset.

Parameters
include_individual (*bool*) – If True, summaries for individual files are included.

Returns
dict - a dictionary with 'Dataset' and 'Individual files' keys.

Notes

- The 'Dataset' value is either a string or a dictionary with the overall summary.
- **The 'Individual files' value is dictionary whose keys are file names and values are** their corresponding summaries.

Users are expected to provide merge_all_info and get_details_dict functions to support this.

HedTypeSummary.**get_text_summary**(*individual_summaries='separate'*)

Return a complete text summary by assembling the individual pieces.

Parameters
individual_summaries (*str*) – One of the values "separate", "consolidated", or "none".

Returns
Complete text summary.

Return type
str

Notes: The options are:

- "none": Just has "Dataset" key.

- "consolidated" Has "Dataset" and "Individual files" keys with the values of each is a string.
- "separate" Has "Dataset" and "Individual files" keys. The values of "Individual files" is a dict.

HedTypeSummary.**get_text_summary_details**(*include_individual=True*)

Return a text summary of the information represented by this summary.

Parameters

include_individual (*bool*) – If True (the default), individual summaries are in "Individual files".

HedTypeSummary.**merge_all_info**()

Create a HedTypeCounts containing the overall dataset HED type summary.

Returns

HedTypeCounts - the overall dataset summary object for HED type summary.

HedTypeSummary.**save**(*save_dir*, *file_formats=['.txt']*, *individual_summaries='separate'*, *task_name=''*)

Save the summaries using the format indicated.

Parameters

- **save_dir** (*str*) – Name of the directory to save the summaries in.
- **file_formats** (*list*) – List of file formats to use for saving.
- **individual_summaries** (*str*) – Save one file or multiple files based on setting.
- **task_name** (*str*) – If this summary corresponds to files from a task, the task_name is used in filename.

HedTypeSummary.**save_visualizations**(*save_dir*, *file_formats=['.svg']*, *individual_summaries='separate'*, *task_name=''*)

Save summary visualizations, if any, using the format indicated.

Parameters

- **save_dir** (*str*) – Name of the directory to save the summaries in.
- **file_formats** (*list*) – List of file formats to use for saving.
- **individual_summaries** (*str*) – Save one file or multiple files based on setting.
- **task_name** (*str*) – If this summary corresponds to files from a task, the task_name is used in filename.

HedTypeSummary.**update_summary**(*new_info*)

Update the summary for a given tabular input file.

Parameters

new_info (*dict*) – A dictionary with the parameters needed to update a summary.

Notes

- The summary needs a "name" str, a "schema", a "df, and a "Sidecar".

HedTypeSummary.**DISPLAY_INDENT = ' '**

HedTypeSummary.**INDIVIDUAL_SUMMARIES_PATH = 'individual_summaries'**

3.4.3.4.20.2 SummarizeHedTypeOp

class SummarizeHedTypeOp(*parameters*)

Summarize a HED type tag in a collection of tabular files.

Required remodeling parameters:

- **summary_name** (*str*): The name of the summary.
- **summary_filename** (*str*): Base filename of the summary.
- **type_tag** (*str*):Type tag to get_summary (e.g. *condition-variable* or *task* tags).

Optional remodeling parameters:

- **append_timecode** (*bool*): If true, the timecode is appended to the base filename when summary is saved.

The purpose of this op is to produce a summary of the occurrences of specified tag. This summary is often used with *condition-variable* to produce a summary of the experimental design.

Methods

SummarizeHedTypeOp.__init__(parameters)	Constructor for the summarize HED type operation.
SummarizeHedTypeOp.do_op(dispatcher, df, name)	Summarize a specified HED type variable such as Condition-variable.
SummarizeHedTypeOp.validate_input_data(...)	Additional validation required of operation parameters not performed by JSON schema validator.

Attributes

SummarizeHedTypeOp.NAME
SummarizeHedTypeOp.PARAMS
SummarizeHedTypeOp.SUMMARY_TYPE

SummarizeHedTypeOp.**__init__**(*parameters*)

Constructor for the summarize HED type operation.

Parameters
parameters (*dict*) – Dictionary with the parameter values for required and optional parameters.

SummarizeHedTypeOp.**do_op**(*dispatcher*, *df*, *name*, *sidecar=None*)

Summarize a specified HED type variable such as Condition-variable.

Parameters

- **dispatcher** (*Dispatcher*) – Manages the operation I/O.
- **df** (*DataFrame*) – The DataFrame to be summarized.
- **name** (*str*) – Unique identifier for the dataframe – often the original file path.
- **sidecar** (*Sidecar or file-like*) – Usually required unless event file has a HED column.

Returns

A copy of df

Return type

DataFrame

Side effect:

Updates the relevant summary.

static SummarizeHedTypeOp.**validate_input_data**(*parameters*)

Additional validation required of operation parameters not performed by JSON schema validator.

SummarizeHedTypeOp.**NAME = 'summarize_hed_type'**

SummarizeHedTypeOp.**PARAMS = {'additionalProperties': False, 'properties': {'append_timecode': {'description': 'If true, the timecode is appended to the base filename so each run has a unique name.', 'type': 'boolean'}, 'summary_filename': {'description': 'Name to use for the summary file name base.', 'type': 'string'}, 'summary_name': {'description': 'Name to use for the summary in titles.', 'type': 'string'}, 'type_tag': {'description': 'Type tag (such as Condition-variable or Task to design summaries for..', 'type': 'string'}}, 'required': ['summary_name', 'summary_filename', 'type_tag'], 'type': 'object'}**

SummarizeHedTypeOp.**SUMMARY_TYPE = 'hed_type_summary'**

3.4.3.4.21 summarize_hed_validation_op

Validate the HED tags in a dataset and report errors.

Classes

HedValidationSummary(sum_op)	Manager for summary of validation issues.
SummarizeHedValidationOp(parameters)	Validate the HED tags in a dataset and report errors.

3.4.3.4.21.1 HedValidationSummary

class HedValidationSummary(*sum_op*)

Manager for summary of validation issues.

Methods

HedValidationSummary.__init__(sum_op)	Constructor for validation issue manager.
HedValidationSummary.dump_summary(filename, ...)	
HedValidationSummary.get_details_dict(...)	Return the summary details from the summary_info.
HedValidationSummary.get_empty_results()	Return an empty results dictionary to use as a template.
HedValidationSummary.get_error_list(error_dict)	Convert errors produced by the HED validation into a list which includes filenames.
HedValidationSummary.get_individual(...[, ...])	Return a dictionary of the individual file summaries.
HedValidationSummary.get_summary([...])	Return a summary dictionary with the information.
HedValidationSummary.get_summary_details([...])	Return a dictionary with the details for individual files and the overall dataset.
HedValidationSummary.get_text_summary([...])	Return a complete text summary by assembling the individual pieces.
HedValidationSummary.get_text_summary_details([...])	Return a text summary of the information represented by this summary.
HedValidationSummary.merge_all_info()	Create a dictionary containing all the errors in the dataset.
HedValidationSummary.save(save_dir[, ...])	Save the summaries using the format indicated.
HedValidationSummary.save_visualizations(...)	Save summary visualizations, if any, using the format indicated.
HedValidationSummary.update_error_location(...)	Updates error information about where an error occurred in sidecar or columnar file.
HedValidationSummary.update_summary(new_info)	Update the summary for a given tabular input file.

Attributes

HedValidationSummary.DISPLAY_INDENT
HedValidationSummary.INDIVIDUAL_SUMMARIES_PATH

HedValidationSummary.**__init__**(*sum_op*)

Constructor for validation issue manager.

Parameters

sum_op (*SummarizeHedValidationOp*) – Operation associated with this summary.

static HedValidationSummary.**dump_summary**(*filename*, *summary*)

HedValidationSummary.**get_details_dict**(*summary_info*)

Return the summary details from the summary_info.

Parameters
summary_info (*dict*) – Dictionary of issues

Returns
Same summary_info as was passed in.

Return type
dict

static HedValidationSummary.**get_empty_results**()

Return an empty results dictionary to use as a template.

Returns
Dictionary template of results info for the validation summary to fill in

Return type
dict

static HedValidationSummary.**get_error_list**(*error_dict*, *count_only=False*)

Convert errors produced by the HED validation into a list which includes filenames.

Parameters

- **error_dict** (*dict*) – Dictionary {filename: error_list} from validation.
- **count_only** (*bool*) – If False (the default), a full list of errors is included otherwise only error counts.

Returns
Error list of form [filenameA, issueA1, issueA2, . . . , filenameB, issueB1, . . .].

Return type
list

HedValidationSummary.**get_individual**(*summary_details*, *separately=True*)

Return a dictionary of the individual file summaries.

Parameters

- **summary_details** (*dict*) – Dictionary of the individual file summaries.
- **separately** (*bool*) – If True (the default), each individual summary has a header for separate output.

HedValidationSummary.**get_summary**(*individual_summaries='separate'*)

Return a summary dictionary with the information.

Parameters
individual_summaries (*str*) – “separate”, “consolidated”, or “none”

Returns
dict - dictionary with “Dataset” and “Individual files” keys.

Notes: The individual_summaries value is processed as follows:

- “separate” individual summaries are to be in separate files.
- “consolidated” means that the individual summaries are in same file as overall summary.
- “none” means that only the overall summary is produced.

HedValidationSummary.**get_summary_details**(*include_individual=True*)

Return a dictionary with the details for individual files and the overall dataset.

Parameters
include_individual (*bool*) – If True, summaries for individual files are included.

Returns
dict - a dictionary with 'Dataset' and 'Individual files' keys.

Notes

- The 'Dataset' value is either a string or a dictionary with the overall summary.
- **The 'Individual files' value is dictionary whose keys are file names and values are** their corresponding summaries.

Users are expected to provide merge_all_info and get_details_dict functions to support this.

HedValidationSummary.**get_text_summary**(*individual_summaries='separate'*)

Return a complete text summary by assembling the individual pieces.

Parameters
individual_summaries (*str*) – One of the values "separate", "consolidated", or "none".

Returns
Complete text summary.

Return type
str

Notes: The options are:

- "none": Just has "Dataset" key.
- "consolidated" Has "Dataset" and "Individual files" keys with the values of each is a string.
- "separate" Has "Dataset" and "Individual files" keys. The values of "Individual files" is a dict.

HedValidationSummary.**get_text_summary_details**(*include_individual=True*)

Return a text summary of the information represented by this summary.

Parameters
include_individual (*bool*) – If True (the default), individual summaries are in "Individual files".

HedValidationSummary.**merge_all_info**()

Create a dictionary containing all the errors in the dataset.

Returns
dict - dictionary of issues organized into sidecar_issues and event_issues.

HedValidationSummary.**save**(*save_dir*, *file_formats=['.txt']*, *individual_summaries='separate'*, *task_name=''*)

Save the summaries using the format indicated.

Parameters

- **save_dir** (*str*) – Name of the directory to save the summaries in.
- **file_formats** (*list*) – List of file formats to use for saving.
- **individual_summaries** (*str*) – Save one file or multiple files based on setting.

- **task_name** (*str*) – If this summary corresponds to files from a task, the task_name is used in filename.

HedValidationSummary.**save_visualizations**(*save_dir*, *file_formats=['.svg']*, *individual_summaries='separate'*, *task_name=''*)

Save summary visualizations, if any, using the format indicated.

Parameters

- **save_dir** (*str*) – Name of the directory to save the summaries in.
- **file_formats** (*list*) – List of file formats to use for saving.
- **individual_summaries** (*str*) – Save one file or multiple files based on setting.
- **task_name** (*str*) – If this summary corresponds to files from a task, the task_name is used in filename.

static HedValidationSummary.**update_error_location**(*error_locations*, *location_name*, *location_key*, *error*)

Updates error information about where an error occurred in sidecar or columnar file.

Parameters

- **error_locations** (*list*) – List of error locations detected so far is this error.
- **location_name** (*str*) – Error location name, for example 'row', 'column', or 'sidecar column'.
- **location_key** (*str*) – Standard key name for this location in the dictionary for an error.
- **error** (*dict*) – Dictionary containing the information about this error.

HedValidationSummary.**update_summary**(*new_info*)

Update the summary for a given tabular input file.

Parameters

new_info (*dict*) – A dictionary with the parameters needed to update a summary.

Notes

- The summary needs a "name" str, a schema, a "df", and a "Sidecar".

HedValidationSummary.**DISPLAY_INDENT = ' '**

HedValidationSummary.**INDIVIDUAL_SUMMARIES_PATH = 'individual_summaries'**

3.4.3.4.21.2 SummarizeHedValidationOp

class SummarizeHedValidationOp(*parameters*)

Validate the HED tags in a dataset and report errors.

Required remodeling parameters:

- **summary_name** (*str*): The name of the summary.
- **summary_filename** (*str*): Base filename of the summary.
- **check_for_warnings** (*bool*): If true include warnings as well as errors.

Optional remodeling parameters:

- **append_timecode** (*bool*): If true, the timecode is appended to the base filename when summary is saved.

The purpose of this op is to produce a summary of the HED validation errors in a file.

Methods

SummarizeHedValidationOp.__init__(parameters)	Constructor for the summarize HED validation operation.
SummarizeHedValidationOp.do_op(dispatcher, ...)	Validate the dataframe with the accompanying sidecar, if any.
SummarizeHedValidationOp.validate_input_data(...)	Additional validation required of operation parameters not performed by JSON schema validator.

Attributes

SummarizeHedValidationOp.NAME
SummarizeHedValidationOp.PARAMS
SummarizeHedValidationOp.SUMMARY_TYPE

SummarizeHedValidationOp.**__init__**(*parameters*)

Constructor for the summarize HED validation operation.

Parameters
parameters (*dict*) – Dictionary with the parameter values for required and optional parameters.

SummarizeHedValidationOp.**do_op**(*dispatcher*, *df*, *name*, *sidecar=None*)

Validate the dataframe with the accompanying sidecar, if any.

Parameters

- **dispatcher** (*Dispatcher*) – Manages the operation I/O.
- **df** (*DataFrame*) – The DataFrame to be validated.
- **name** (*str*) – Unique identifier for the dataframe – often the original file path.
- **sidecar** (*Sidecar or file-like*) – Usually needed unless only HED tags in HED column of event file.

Returns
A copy of df

Return type
DataFrame

Side effect:
Updates the relevant summary.

static SummarizeHedValidationOp.**validate_input_data**(*parameters*)

Additional validation required of operation parameters not performed by JSON schema validator.

SummarizeHedValidationOp.**NAME** = 'summarize_hed_validation'

SummarizeHedValidationOp.**PARAMS** = {'additionalProperties': False, 'properties': {'append_timecode': {'description': 'If true, the timecode is appended to the base filename so each run has a unique name.', 'type': 'boolean'}, 'check_for_warnings': {'description': 'If true warnings as well as errors are reported.', 'type': 'boolean'}, 'summary_filename': {'description': 'Name to use for the summary file name base.', 'type': 'string'}, 'summary_name': {'description': 'Name to use for the summary in titles.', 'type': 'string'}}, 'required': ['summary_name', 'summary_filename', 'check_for_warnings'], 'type': 'object'}

SummarizeHedValidationOp.**SUMMARY_TYPE** = 'hed_validation'

3.4.3.4.22 summarize_sidecar_from_events_op

Create a JSON sidecar from column values in a collection of tabular files.

Classes

EventsToSidecarSummary(sum_op)	Manager for events to sidecar generation.
SummarizeSidecarFromEventsOp(parameters)	Create a JSON sidecar from column values in a collection of tabular files.

3.4.3.4.22.1 EventsToSidecarSummary

class EventsToSidecarSummary(*sum_op*)

Manager for events to sidecar generation.

Methods

EventsToSidecarSummary.__init__(sum_op)	Constructor for events to sidecar manager.
EventsToSidecarSummary.dump_summary(...)	
EventsToSidecarSummary.get_details_dict(...)	Return the summary-specific information.
EventsToSidecarSummary.get_individual(...[, ...])	Return a dictionary of the individual file summaries.
EventsToSidecarSummary.get_summary([...])	Return a summary dictionary with the information.
EventsToSidecarSummary.get_summary_details([...])	Return a dictionary with the details for individual files and the overall dataset.
EventsToSidecarSummary.get_text_summary([...])	Return a complete text summary by assembling the individual pieces.
EventsToSidecarSummary.get_text_summary_details([...])	Return a text summary of the information represented by this summary.
EventsToSidecarSummary.merge_all_info()	Merge summary information from all the files.
EventsToSidecarSummary.save(save_dir[, ...])	Save the summaries using the format indicated.
EventsToSidecarSummary.save_visualizations(...)	Save summary visualizations, if any, using the format indicated.
EventsToSidecarSummary.update_summary(new_info)	Update the summary for a given tabular input file.
EventsToSidecarSummary.validate_input_data(...)	

Attributes

EventsToSidecarSummary.DISPLAY_INDENT
EventsToSidecarSummary.INDIVIDUAL_SUMMARIES_PATH

EventsToSidecarSummary.**__init__**(*sum_op*)

Constructor for events to sidecar manager.

Parameters
sum_op (*BaseOp*) – Operation associated with this summary.

static EventsToSidecarSummary.**dump_summary**(*filename*, *summary*)

EventsToSidecarSummary.**get_details_dict**(*summary_info*)

Return the summary-specific information.

Parameters
summary_info (*TabularSummary*) – Summary to return info from.

Returns
Standardized details dictionary extracted from the summary information.

Return type
dict

Notes

Abstract method be implemented by each individual context summary.

EventsToSidecarSummary.**get_individual**(*summary_details*, *separately=True*)

Return a dictionary of the individual file summaries.

Parameters

- **summary_details** (*dict*) – Dictionary of the individual file summaries.
- **separately** (*bool*) – If True (the default), each individual summary has a header for separate output.

EventsToSidecarSummary.**get_summary**(*individual_summaries='separate'*)

Return a summary dictionary with the information.

Parameters
individual_summaries (*str*) – "separate", "consolidated", or "none"

Returns
dict - dictionary with "Dataset" and "Individual files" keys.

Notes: The individual_summaries value is processed as follows:

- "separate" individual summaries are to be in separate files.
- "consolidated" means that the individual summaries are in same file as overall summary.
- "none" means that only the overall summary is produced.

EventsToSidecarSummary.**get_summary_details**(*include_individual=True*)

Return a dictionary with the details for individual files and the overall dataset.

Parameters
include_individual (*bool*) – If True, summaries for individual files are included.

Returns
dict - a dictionary with 'Dataset' and 'Individual files' keys.

Notes

- The 'Dataset' value is either a string or a dictionary with the overall summary.
- **The 'Individual files' value is dictionary whose keys are file names and values are** their corresponding summaries.

Users are expected to provide merge_all_info and get_details_dict functions to support this.

EventsToSidecarSummary.**get_text_summary**(*individual_summaries='separate'*)

Return a complete text summary by assembling the individual pieces.

Parameters
individual_summaries (*str*) – One of the values "separate", "consolidated", or "none".

Returns
Complete text summary.

Return type
str

Notes: The options are:

- "none": Just has "Dataset" key.
- "consolidated" Has "Dataset" and "Individual files" keys with the values of each is a string.
- "separate" Has "Dataset" and "Individual files" keys. The values of "Individual files" is a dict.

EventsToSidecarSummary.**get_text_summary_details**(*include_individual=True*)

Return a text summary of the information represented by this summary.

Parameters

include_individual (*bool*) – If True (the default), individual summaries are in "Individual files".

EventsToSidecarSummary.**merge_all_info**()

Merge summary information from all the files.

Returns

Consolidated summary of information.

Return type

TabularSummary

EventsToSidecarSummary.**save**(*save_dir*, *file_formats=['.txt']*, *individual_summaries='separate'*, *task_name=''*)

Save the summaries using the format indicated.

Parameters

- **save_dir** (*str*) – Name of the directory to save the summaries in.
- **file_formats** (*list*) – List of file formats to use for saving.
- **individual_summaries** (*str*) – Save one file or multiple files based on setting.
- **task_name** (*str*) – If this summary corresponds to files from a task, the task_name is used in filename.

EventsToSidecarSummary.**save_visualizations**(*save_dir*, *file_formats=['.svg']*, *individual_summaries='separate'*, *task_name=''*)

Save summary visualizations, if any, using the format indicated.

Parameters

- **save_dir** (*str*) – Name of the directory to save the summaries in.
- **file_formats** (*list*) – List of file formats to use for saving.
- **individual_summaries** (*str*) – Save one file or multiple files based on setting.
- **task_name** (*str*) – If this summary corresponds to files from a task, the task_name is used in filename.

EventsToSidecarSummary.**update_summary**(*new_info*)

Update the summary for a given tabular input file.

Parameters

new_info (*dict*) – A dictionary with the parameters needed to update a summary.

Notes

- The summary needs a "name" str and a "df".

static EventsToSidecarSummary.**validate_input_data**(*parameters*)

EventsToSidecarSummary.**DISPLAY_INDENT = ' '**

EventsToSidecarSummary.**INDIVIDUAL_SUMMARIES_PATH = 'individual_summaries'**

3.4.3.4.22.2 SummarizeSidecarFromEventsOp

class SummarizeSidecarFromEventsOp(*parameters*)

Create a JSON sidecar from column values in a collection of tabular files.

Required remodeling parameters:

- **summary_name** (*str*): The name of the summary.
- **summary_filename** (*str*): Base filename of the summary.

Optional remodeling parameters:

- **append_timecode** (*bool*):
- **skip_columns** (*list*): Names of columns to skip in the summary.
- **value_columns** (*list*): Names of columns to treat as value columns rather than categorical columns.

The purpose is to produce a JSON sidecar template for annotating a dataset with HED tags.

Methods

SummarizeSidecarFromEventsOp.__init__(parameters)	Constructor for summarize sidecar from events operation.
SummarizeSidecarFromEventsOp.do_op(...[, ...])	Extract a sidecar from events file.
SummarizeSidecarFromEventsOp.validate_input_data(...)	Additional validation required of operation parameters not performed by JSON schema validator.

Attributes

SummarizeSidecarFromEventsOp.NAME
SummarizeSidecarFromEventsOp.PARAMS
SummarizeSidecarFromEventsOp.SUMMARY_TYPE

SummarizeSidecarFromEventsOp.**__init__**(*parameters*)

Constructor for summarize sidecar from events operation.

Parameters
parameters (*dict*) – Dictionary with the parameter values for required and optional parameters.

SummarizeSidecarFromEvents0p.**do_op**(*dispatcher*, *df*, *name*, *sidecar=None*)

Extract a sidecar from events file.

Parameters

- **dispatcher** (*Dispatcher*) – The dispatcher object for managing the operations.
- **df** (*DataFrame*) – The tabular file to be remodeled.
- **name** (*str*) – Unique identifier for the dataframe – often the original file path.
- **sidecar** (*Sidecar or file-like*) – Not needed for this operation.

Returns
A copy of df.

Return type
DataFrame

Side effect:
Updates the associated summary if applicable.

static SummarizeSidecarFromEvents0p.**validate_input_data**(*parameters*)

Additional validation required of operation parameters not performed by JSON schema validator.

SummarizeSidecarFromEvents0p.**NAME = 'summarize_sidecar_from_events'**

SummarizeSidecarFromEvents0p.**PARAMS = {'additionalProperties': False, 'properties': {'append_timecode': {'type': 'boolean'}, 'skip_columns': {'description': 'List of columns to skip in generating the sidecar.', 'items': {'type': 'string'}, 'minItems': 1, 'type': 'array', 'uniqueItems': True}, 'summary_filename': {'description': 'Name to use for the summary file name base.', 'type': 'string'}, 'summary_name': {'description': 'Name to use for the summary in titles.', 'type': 'string'}, 'value_columns': {'description': 'List of columns to provide a single annotation with placeholder for the values.', 'items': {'type': 'string'}, 'minItems': 1, 'type': 'array', 'uniqueItems': True}}, 'required': ['summary_name', 'summary_filename'], 'type': 'object'}**

SummarizeSidecarFromEvents0p.**SUMMARY_TYPE = 'events_to_sidecar'**

3.4.3.4.23 valid_operations

The valid operations for the remodeling tools.

3.4.3.5 remodeler_validator

Validator for remodeler input files.

Classes

RemodelerValidator()	Validator for remodeler input files.

3.4.3.5.1 RemodelerValidator

class RemodelerValidator

Validator for remodeler input files.

Methods

RemodelerValidator.__init__()	Constructor for remodeler Validator.
RemodelerValidator.validate(operations)	Validate remodeler operations against the json schema specification and specific op requirements.

Attributes

RemodelerValidator.BASE_ARRAY
RemodelerValidator.MESSAGE_STRINGS
RemodelerValidator.OPERATION_DICT
RemodelerValidator.PARAMETER_SPECIFICATION_TEMPLATE

RemodelerValidator.**__init__**()

Constructor for remodeler Validator.

RemodelerValidator.**validate**(*operations*)

Validate remodeler operations against the json schema specification and specific op requirements.

Parameters

operations (*list*) – Dictionary with input operations to run through the remodeler.

Returns

List with the error messages for errors identified by the validator.

Return type

list

RemodelerValidator.**BASE_ARRAY = {'items': {}, 'minItems': 1, 'type': 'array'}**

RemodelerValidator.MESSAGE_STRINGS = {'0': {'minItems': 'There are no operations defined. Specify at least 1 operation for the remodeler to execute.', 'type': 'Operations must be contained in a list or array. This is also true for a single operation.'}, '1': {'additionalProperties': "Operation dictionary {operation_index} contains an unexpected field '{added_property}'. Every operation dictionary must specify the type of operation, a description, and the operation parameters.", 'required': "Operation dictionary {operation_index} is missing '{missing_value}'. Every operation dictionary must specify the type of operation, a description, and the operation parameters.", 'type': 'Each operation must be defined in a dictionary: {instance} is not a dictionary object.'}, '2': {'additionalProperties': "Operation {operation_index}: Operation parameters for {operation_name} contain an unexpected field '{added_property}'.", 'dependentRequired': 'Operation {operation_index}: The parameter {missing_value} is missing: {missing_value} is a required parameter of {operation_name} when {dependent_on} is specified.', 'enum': '{instance} is not a known remodeler operation. See the documentation for valid operations.', 'required': 'Operation {operation_index}: The parameter {missing_value} is missing. {missing_value} is a required parameter of {operation_name}.', 'type': 'Operation {operation_index}: {instance} is not a {validator_value}. {operation_field} should be of type {validator_value}.'}, 'more': {'additionalProperties': "Operation {operation_index}: Operation parameters for {parameter_path} contain an unexpected field '{added_property}'.", 'enum': 'Operation {operation_index}: Operation parameter {parameter_path} in the {operation_name} operation contains and unexpected value. Value should be one of {validator_value}.', 'minItems': 'Operation {operation_index}: The list in {parameter_path} in the {operation_name} operation should have at least {validator_value} item(s).', 'minProperties': 'Operation {operation_index}: The dictionary in {parameter_path} in the {operation_name} operation should have at least {validator_value} key(s).', 'required': 'Operation {operation_index}: The field {missing_value} is missing in {parameter_path}. {missing_value} is a required parameter of {parameter_path}.', 'type': 'Operation {operation_index}: The value of {parameter_path} in the {operation_name} operation should be {validator_value}. {instance} is not a {validator_value}.', 'uniqueItems': 'Operation {operation_index}: The list in {parameter_path} in the {operation_name} operation should only contain unique items.'}}

RemodelerValidator.OPERATION_DICT = {'additionalProperties': False, 'allOf': [], 'properties': {'description': {'type': 'string'}, 'operation': {'default': 'convert_columns', 'enum': [], 'type': 'string'}, 'parameters': {'properties': {}, 'type': 'object'}}, 'required': ['operation', 'description', 'parameters'], 'type': 'object'}

RemodelerValidator.PARAMETER_SPECIFICATION_TEMPLATE = {'if': {'properties': {'operation': {'const': ''}}, 'required': ['operation']}, 'then': {'properties': {'parameters': {}}}}

3.4.4 util

Data and file handling utilities.

Modules

hed.tools.util.data_util	Data handling utilities involving dataframes.
hed.tools.util.hed_logger	Logger class with messages organized by key
hed.tools.util.io_util	Utilities for generating and handling file names.
hed.tools.util.schema_util	Utilities

3.4.4.1 data_util

Data handling utilities involving dataframes.

Functions

add_columns(df, column_list[, value])	Add specified columns to df if not there.
check_match(ds1, ds2[, numeric])	Check two Pandas data series have the same values.
delete_columns(df, column_list)	Delete the specified columns from a dataframe.
delete_rows_by_column(df, value[, column_list])	Delete rows where columns have this value.
get_eligible_values(values, values_included)	Return a list of the items from values that are in values_included or None if no values_included.
get_key_hash(key_tuple)	Calculate a hash key for tuple of values.
get_new_dataframe(data)	Get a new dataframe representing a tsv file.
get_row_hash(row, key_list)	Get a hash key from key column values for row.
get_value_dict(tsv_path[, key_col, value_col])	Get a dictionary of two columns of a dataframe.
make_info_dataframe(col_info, selected_col)	Get a dataframe from selected columns.
reorder_columns(data, col_order[, skip_missing])	Create a new dataframe with columns reordered.
replace_na(df)	Replace (in place) the n/a with np.nan taking care of categorical columns.
replace_values(df[, values, replace_value, ...])	Replace string values in specified columns.
separate_values(values, target_values)	Get target values from the target_values list.

add_columns(*df*, *column_list*, *value='n/a'*)

Add specified columns to df if not there.

Parameters

- **df** (*DataFrame*) – Pandas dataframe.
- **column_list** (*list*) – List of columns to append to the dataframe.
- **value** (*str*) – Default fill value for the column.

check_match(*ds1*, *ds2*, *numeric=False*)

Check two Pandas data series have the same values.

Parameters

- **ds1** (*DataSeries*) – Pandas data series to check.
- **ds2** (*DataSeries*) – Pandas data series to check.

- **numeric** (*bool*) – If True, treat as numeric and do close-to comparison.

Returns
Error messages indicating the mismatch or empty if the series match.

Return type
list

delete_columns(*df*, *column_list*)

Delete the specified columns from a dataframe.

Parameters

- **df** (*DataFrame*) – Pandas dataframe from which to delete columns.
- **column_list** (*list*) – List of candidate column names for deletion.

Notes

- The deletion of columns is done in place.
- This does not raise an error if df does not have a column in the list.

delete_rows_by_column(*df*, *value*, *column_list=None*)

Delete rows where columns have this value.

Parameters

- **df** (*DataFrame*) – Pandas dataframe from which to delete rows.
- **value** (*str*) – Specified value to indicate row should be deleted.
- **column_list** (*list*) – List of columns to search for value.

Notes

- All values are converted to string before testing.
- Deletion is done in place.

get_eligible_values(*values*, *values_included*)

Return a list of the items from values that are in values_included or None if no values_included.

Parameters

- **values** (*list*) – List of strings against which to test.
- **values_included** (*list*) – List of items to be selected from values if they are present.

Returns
list of selected values or None if values_included is empty or None.

Return type
list

get_key_hash(*key_tuple*)

Calculate a hash key for tuple of values.

Parameters
key_tuple (*tuple, list*) – The key values in the correct order for lookup.

Returns
A hash key for the tuple.

Return type
int

get_new_dataframe(*data*)

Get a new dataframe representing a tsv file.

Parameters
data (*DataFrame or str*) – DataFrame or filename representing a tsv file.

Returns

A dataframe containing the contents of the tsv file or if data was
a DataFrame to start with, a new copy of the DataFrame.

Return type
DataFrame

Raises
HedFileError –

- A filename is given, and it cannot be read into a Dataframe.

get_row_hash(*row*, *key_list*)

Get a hash key from key column values for row.

Parameters

- **row** (*DataSeries*) –
- **key_list** (*list*) –

Returns
Hash key constructed from the entries of row in the columns specified by key_list.

Return type
str

Raises
HedFileError –

- If row doesn't have all the columns in key_list HedFileError is raised.

get_value_dict(*tsv_path*, *key_col='file_basename'*, *value_col='sampling_rate'*)

Get a dictionary of two columns of a dataframe.

Parameters

- **tsv_path** (*str*) – Path to a tsv file with a header row to be read into a DataFrame.
- **key_col** (*str*) – Name of the column which should be the key.
- **value_col** (*str*) – Name of the column which should be the value.

Returns
Dictionary with key_col values as the keys and the corresponding value_col values as the values.

Return type
dict

Raises
HedFileError –

- When tsv_path does not correspond to a file that can be read into a DataFrame.

make_info_dataframe(*col_info*, *selected_col*)

Get a dataframe from selected columns.

Parameters

- **col_info** (*dict*) – Dictionary of dictionaries of column values and counts.
- **selected_col** (*str*) – Name of the column used as top level key for col_info.

Returns

A two-column dataframe with first column containing values from the
dictionary whose key is selected_col and whose second column are the corresponding counts. The returned value is None if selected_col is not a top-level key in col_info.

Return type
dataframe

reorder_columns(*data*, *col_order*, *skip_missing=True*)

Create a new dataframe with columns reordered.

Parameters

- **data** (*DataFrame, str*) – Dataframe or filename of dataframe whose columns are to be reordered.
- **col_order** (*list*) – List of column names in desired order.
- **skip_missing** (*bool*) – If true, col_order columns missing from data are skipped, otherwise error.

Returns
A new reordered dataframe.

Return type
DataFrame

Raises
HedFileError –

- If col_order contains columns not in data and skip_missing is False.
- If data corresponds to a filename from which a dataframe cannot be created.

replace_na(*df*)

Replace (in place) the n/a with np.nan taking care of categorical columns.

replace_values(*df*, *values=None*, *replace_value='n/a'*, *column_list=None*)

Replace string values in specified columns.

Parameters

- **df** (*DataFrame*) – Dataframe whose values will be replaced.
- **values** (*list, None*) – List of strings to replace. If None, only empty strings are replaced.
- **replace_value** (*str*) – String replacement value.
- **column_list** (*list, None*) – List of columns in which to do replacement. If None all columns are processed.

Returns
number of values replaced.

Return type
int

separate_values(*values*, *target_values*)

Get target values from the target_values list.

Parameters

- **values** (*list*) – List of values to be tested.
- **target_values** – List of desired values.

3.4.4.2 hed_logger

Logger class with messages organized by key

Classes

HedLogger([name])	Log status messages organized by key.

3.4.4.2.1 HedLogger

class HedLogger(*name=None*)

Log status messages organized by key.

Methods

HedLogger.__init__([name])	Constructor for the HED logger.
HedLogger.add(key, msg[, level, also_print])	Add an entry to this log.
HedLogger.get_log(key)	Get all the log entries stored under the key.
HedLogger.get_log_keys()	Return a list of keys for this log.
HedLogger.get_log_string([level])	Return the log as a string, with entries separated by newlines.

Attributes

HedLogger.**__init__**(*name=None*)

Constructor for the HED logger.

Parameters

name (*str*) – Identifying name of the logger.

HedLogger.**add**(*key*, *msg*, *level=''*, *also_print=False*)

Add an entry to this log.

Parameters

- **key** (*str*) – Key used to organize log messages.
- **msg** (*str*) – Message to log.
- **level** (*str*) – Level of importance for filtering messages.

- **also_print** (*bool*) – If False (the default) nothing is output, otherwise the log entry output to stdout.

HedLogger.**get_log**(*key*)

Get all the log entries stored under the key.

Parameters
key (*str*) – The key whose log messages are retrieved.

Returns
List of log entries associated with this key.

Return type
list

HedLogger.**get_log_keys**()

Return a list of keys for this log.

Returns
list of organizational keys for this log.

Return type
list

HedLogger.**get_log_string**(*level=None*)

Return the log as a string, with entries separated by newlines.

Parameters
level (*str or None*) – Include only the entries from this level. If None, do all.

Returns
The log as a string separated by newlines.

Return type
str

3.4.4.3 io_util

Utilities for generating and handling file names.

Functions

check_filename(test_file[, name_prefix, ...])	Return True if correct extension, suffix, and prefix.
clean_filename(filename)	Replace invalid characters with under-bars.
extract_suffix_path(path, prefix_path)	Return the suffix of path after prefix path has been removed.
get_allowed(value[, allowed_values, starts_with])	Return the portion of the value that matches a value in allowed_values or None if no match.
get_alphanumeric_path(pathname[, replace_char])	Replace sequences of non-alphanumeric characters in string (usually a path) with specified character.
get_basename(file_path)	
get_file_list(root_path[, name_prefix, ...])	Return paths satisfying various conditions.
get_filtered_by_element(file_list, elements)	Filter a file list by whether the base names have a substring matching any of the members of elements.
get_filtered_list(file_list[, name_prefix, ...])	Get list of filenames satisfying the criteria.
get_full_extension(filename)	Return the full extension of a file, including the period.
get_path_components(root_path, this_path)	Get a list of the remaining components after root path.
get_task_dict(files)	Return a dictionary of the tasks that appear in the file names of a list of files.
get_task_from_file(file_path)	Returns the task name entity from a BIDS-type file path.
get_timestamp()	Return a timestamp string suitable for using in filenames.
get_unique_suffixes(file_paths[, extensions])	
make_path(root_path, sub_path, filename)	Get path for a file, verifying all components exist.
separate_by_ext(file_paths)	Separate a list of files into tsv and json files.

check_filename(*test_file*, *name_prefix=None*, *name_suffix=None*, *extensions=None*)

Return True if correct extension, suffix, and prefix.

Parameters

- **test_file** (*str*) – Path of filename to test.
- **name_prefix** (*list, str, None*) – An optional name_prefix or list of prefixes to accept for the base filename.
- **name_suffix** (*list, str, None*) – An optional name_suffix or list of suffixes to accept for the base file name.
- **extensions** (*list, str, None*) – An optional extension or list of extensions to accept for the extensions.

Returns

True if file has the appropriate format.

Return type

bool

Notes

- Everything is converted to lower case prior to testing so this test should be case-insensitive.
- None indicates that all are accepted.

clean_filename(*filename*)

Replace invalid characters with under-bars.

Parameters
filename (*str*) – source filename.

Returns
The filename with anything but alphanumeric, period, hyphens, and under-bars removed.

Return type
str

extract_suffix_path(*path*, *prefix_path*)

Return the suffix of path after prefix path has been removed.

Parameters

- **path** (*str*) –
- **prefix_path** (*str*) –

Returns
Suffix path.

Return type
str

Notes

- This function is useful for creating files within BIDS datasets.

get_allowed(*value*, *allowed_values=None*, *starts_with=True*)

Return the portion of the value that matches a value in allowed_values or None if no match.

Parameters

- **value** (*str*) – value to be matched.
- **allowed_values** (*list, str, or None*) – Values to match.
- **starts_with** (*bool*) – If True match is done at beginning of string, otherwise the end.

Returns
portion of value that matches the various allowed_values.

Return type
str or list

Notes

- match is done in lower case.

get_alphanumeric_path(*pathname*, *replace_char='_'*)

Replace sequences of non-alphanumeric characters in string (usually a path) with specified character.

Parameters

- **pathname** (*str*) – A string usually representing a pathname, but could be any string.
- **replace_char** (*str*) – Replacement character(s).

Returns
New string with characters replaced.

Return type
str

get_basename(*file_path*)

get_file_list(*root_path*, *name_prefix=None*, *name_suffix=None*, *extensions=None*, *exclude_dirs=None*)

Return paths satisfying various conditions.

Parameters

- **root_path** (*str*) – Full path of the directory tree to be traversed (no ending slash).
- **name_prefix** (*list, str, None*) – An optional prefix for the base filename.
- **name_suffix** (*list, str, None*) – An optional suffix for the base filename.
- **extensions** (*list, None*) – A list of extensions to be selected.
- **exclude_dirs** (*list, None*) – A list of paths to be excluded.

Returns
The full paths.

Return type
list

Notes: Exclude directories are paths relative to the root path.

get_filtered_by_element(*file_list*, *elements*)

Filter a file list by whether the base names have a substring matching any of the members of elements.

Parameters

- **file_list** (*list*) – List of file paths to be filtered.
- **elements** (*list*) – List of strings to use as filename filters.

Returns
The list only containing file paths whose filenames match a filter.

Return type
list

get_filtered_list(*file_list*, *name_prefix=None*, *name_suffix=None*, *extensions=None*)

Get list of filenames satisfying the criteria.

Everything is converted to lower case prior to testing so this test should be case-insensitive.

Parameters

- **file_list** (*list*) – List of files to test.
- **name_prefix** (*str*) – Optional name_prefix for the base filename.
- **name_suffix** (*str*) – Optional name_suffix for the base filename.
- **extensions** – Optional list of file extensions (allows two periods (.tsv.gz)).

get_full_extension(*filename*)

Return the full extension of a file, including the period.

Parameters
filename (*str*) – The filename to be parsed.

Returns
(File name without extension, full extension)

Return type
(str, str)

get_path_components(*root_path*, *this_path*)

Get a list of the remaining components after root path.

Parameters

- **root_path** (*str*) – A path (no trailing separator).
- **this_path** (*str*) – The path of a file or directory descendant of root_path.

Returns
A list with the remaining elements directory components to the file.

Return type
list or None

Notes: this_path must be a descendant of root_path.

get_task_dict(*files*)

Return a dictionary of the tasks that appear in the file names of a list of files.

Parameters
files (*list*) – List of filenames to be separated by task.

Returns
dictionary of filenames keyed by task name.

Return type
dict

get_task_from_file(*file_path*)

Returns the task name entity from a BIDS-type file path.

Parameters
file_path (*str*) – File path.

Returns
The task name or an empty string.

Return type
str

get_timestamp()

Return a timestamp string suitable for using in filenames.

Returns
Represents the current time.

Return type
str

get_unique_suffixes(*file_paths*, *extensions=['.json', '.tsv']*)

make_path(*root_path*, *sub_path*, *filename*)
Get path for a file, verifying all components exist.

Parameters

- **root_path** (*str*) – path of the root directory.
- **sub_path** (*str*) – sub-path relative to the root directory.
- **filename** (*str*) – filename of the file.

Returns
A valid realpath for the specified file.

Return type
str

Notes: This function is useful for creating files within BIDS datasets.

separate_by_ext(*file_paths*)
Separate a list of files into tsv and json files.

Parameters
file_paths (*list*) – A list of file paths.

Returns
key is extension and value is list of files with that extension.

Return type
dict

3.4.4.4 schema_util

Utilities

Functions

flatten_schema(hed_schema[, skip_non_tag])	Returns a 3-column dataframe representing a schema.

flatten_schema(*hed_schema*, *skip_non_tag=False*)
Returns a 3-column dataframe representing a schema.

Parameters

- **hed_schema** (*HedSchema*) – the schema to flatten
- **skip_non_tag** (*bool*) – Skips all sections except tag

Returns
Represents a HED schema in flattened form.

Return type
DataFrame

3.4.5 visualization

Visualization tools for HED.

Modules

hed.tools.visualization.tag_word_cloud	Utilities for creating a word cloud.
hed.tools.visualization.word_cloud_util	Support utilities for word cloud generation.

3.4.5.1 tag_word_cloud

Utilities for creating a word cloud.

Functions

create_wordcloud(word_dict[, mask_path, ...])	Takes a word dict and returns a generated word cloud object.
load_and_resize_mask(mask_path[, width, height])	Load a mask image and resize it according to given dimensions.
word_cloud_to_svg(wc)	Return a WordCould as an SVG string.

create_wordcloud(*word_dict*, *mask_path=None*, *background_color=None*, *width=400*, *height=300*, ***kwargs*)

Takes a word dict and returns a generated word cloud object.

Parameters

- **word_dict** (*dict*) – words and their frequencies
- **mask_path** (*str or None*) – The path of the mask file
- **background_color** (*str or None*) – If None, transparent background.
- **width** (*int*) – width in pixels.
- **height** (*int*) – height in pixels.
- **kwargs** (*kwargs*) – Any other parameters WordCloud accepts, overrides default values where relevant.

Returns

The generated cloud. (Use .to_file to save it out as an image.)

Return type

WordCloud

Raises

ValueError – An empty dictionary was passed

load_and_resize_mask(*mask_path*, *width=None*, *height=None*)

Load a mask image and resize it according to given dimensions.

The image is resized maintaining aspect ratio if only width or height is provided.

Returns None if no mask_path.

Parameters

- **mask_path** (*str*) – The path to the mask image file.
- **width** (*int, optional*) – The desired width of the resized image. If only width is provided, the image is scaled to maintain its original aspect ratio. Defaults to None.
- **height** (*int, optional*) – The desired height of the resized image. If only height is provided, the image is scaled to maintain its original aspect ratio. Defaults to None.

Returns
The loaded and processed mask image as a numpy array with binary values (0 or 255).

Return type
numpy.ndarray

word_cloud_to_svg(*wc*)

Return a WordCould as an SVG string.

Parameters
wc (*WordCloud*) – the word cloud object.

Returns
The svg for the word cloud.

Return type
svg_string (str)

3.4.5.2 word_cloud_util

Support utilities for word cloud generation.

Functions

default_color_func(word, font_size, ...[, ...])	Update the current color fraction and wrap around if necessary.
generate_contour_svg(wc, width, height)	Generate an SVG contour mask based on a word cloud object and dimensions.
random_color_darker([random_state])	Random color generation function.

default_color_func(*word*, *font_size*, *position*, *orientation*, *random_state=None*, ***kwargs*)

Update the current color fraction and wrap around if necessary.

generate_contour_svg(*wc*, *width*, *height*)

Generate an SVG contour mask based on a word cloud object and dimensions.

Parameters

- **wc** (*WordCloud*) – The word cloud object.
- **width** (*int*) – SVG image width in pixels.
- **height** (*int*) – SVG image height in pixels.

Returns
SVG point list for the contour mask, or empty string if not generated.

Return type
str

random_color_darker(*random_state=None*)

Random color generation function.

Parameters
random_state (*Random or None*) – Previous state of random generation for next color generation.

Returns
Represents a hue, saturation, and lightness.

Return type
str

Classes

ColormapColorFunc([colormap, color_range, ...])	Represents a colormap.

3.4.5.2.1 ColormapColorFunc

class ColormapColorFunc(*colormap='nipy_spectral'*, *color_range=(0.0, 0.5)*, *color_step_range=(0.15, 0.25)*)

Represents a colormap.

Methods

ColormapColorFunc.__init__([colormap, ...])	Initialize a word cloud color generator.
ColormapColorFunc.color_func(word, ...[, ...])	Update the current color fraction and wrap around if necessary.

Attributes

ColormapColorFunc.**__init__**(*colormap='nipy_spectral'*, *color_range=(0.0, 0.5)*, *color_step_range=(0.15, 0.25)*)

Initialize a word cloud color generator.

Parameters

- **colormap** (*str, optional*) – The name of the matplotlib colormap to use for generating colors. Defaults to 'nipy_spectral'.
- **color_range** (*tuple of float, optional*) – A tuple containing the minimum and maximum values to use from the colormap. Defaults to (0.0, 0.5).
- **color_step_range** (*tuple of float, optional*) – A tuple containing the minimum and maximum values to step through the colormap. Defaults to (0.15, 0.25). This is the speed at which it goes through the range chosen. .25 means it will go through 1/4 of the range each pick.

ColormapColorFunc.**color_func**(*word*, *font_size*, *position*, *orientation*, *random_state=None*, ***kwargs*)

Update the current color fraction and wrap around if necessary.

3.5 validator

Validation of HED tags.

Modules

hed.validator.def_validator	Validates of Def, Def-expand and Temporal groups.
hed.validator.hed_validator	Top level validation of HED strings.
hed.validator.onset_validator	Validates the onset/offset conditions.
hed.validator.reserved_checker	
hed.validator.sidecar_validator	Validates sidecars.
hed.validator.spreadsheet_validator	Validates spreadsheet tabular data.
hed.validator.util	Validation of HED tags.

3.5.1 def_validator

Validates of Def, Def-expand and Temporal groups.

Classes

`DefValidator`([def_dicts, hed_schema])	Validates Def/ and Def-expand/, as well as Temporal groups: Onset, Inset, and Offset

3.5.1.1 DefValidator

class DefValidator(*def_dicts=None*, *hed_schema=None*)

Validates Def/ and Def-expand/, as well as Temporal groups: Onset, Inset, and Offset

Methods

DefValidator.__init__([def_dicts, hed_schema])	Initialize for definitions in HED strings.
DefValidator.add_definitions(def_dicts[, ...])	Add definitions from dict(s) or strings(s) to this dict.
DefValidator.check_for_definitions(...[, ...])	Check string for definition tags, adding them to self.
DefValidator.get(def_name)	Get the definition entry for the definition name.
DefValidator.get_as_strings(def_dict)	Convert the entries to strings of the contents
DefValidator.get_definition_entry(def_tag)	Get the entry for a given def tag.
DefValidator.items()	Return the dictionary of definitions.
DefValidator.validate_def_tags(hed_string_obj)	Validate Def/Def-Expand tags.
DefValidator.validate_def_value_units(...[, ...])	Equivalent to HedValidator.validate_units for the special case of a Def or Def-expand tag
DefValidator.validate_onset_offset(...)	Validate onset/offset

Attributes

DefValidator.issues	Return issues about duplicate definitions.

DefValidator.**__init__**(*def_dicts=None*, *hed_schema=None*)

Initialize for definitions in HED strings.

Parameters

- **def_dicts** (*list or DefinitionDict or str*) – DefinitionDicts containing the definitions to pass to baseclass
- **hed_schema** (*HedSchema or None*) – Required if passing strings or lists of strings, unused otherwise.

DefValidator.**add_definitions**(*def_dicts*, *hed_schema=None*)

Add definitions from dict(s) or strings(s) to this dict.

Parameters

- **def_dicts** (*list, DefinitionDict, dict, or str*) – DefinitionDict or list of DefinitionDicts/strings/dicts whose definitions should be added.
- **hed_schema** (*HedSchema or None*) – Required if passing strings or lists of strings, unused otherwise.

Note - dict form expects DefinitionEntries in the same form as a DefinitionDict

Note - str or list of strings will parse the strings using the hed_schema. Note - You can mix and match types, eg [DefinitionDict, str, list of str] would be valid input.

Raises

TypeError –

- Bad type passed as def_dicts.

DefValidator.**check_for_definitions**(*hed_string_obj*, *error_handler=None*)

Check string for definition tags, adding them to self.

Parameters

- **hed_string_obj** (*HedString*) – A single HED string to gather definitions from.
- **error_handler** (*ErrorHandler or None*) – Error context used to identify where definitions are found.

Returns

List of issues encountered in checking for definitions. Each issue is a dictionary.

Return type

list

DefValidator.**get**(*def_name*)

Get the definition entry for the definition name.

Not case-sensitive

Parameters

def_name (*str*) – Name of the definition to retrieve.

Returns
Definition entry for the requested definition.

Return type
DefinitionEntry

static DefValidator.**get_as_strings**(*def_dict*)

Convert the entries to strings of the contents

Parameters
def_dict (*DefinitionDict or dict*) – A dict of definitions

Returns
definition name and contents

Return type
dict(str)

DefValidator.**get_definition_entry**(*def_tag*)

Get the entry for a given def tag.

Does not validate at all.

Parameters
def_tag (*HedTag*) – Source HED tag that may be a Def or Def-expand tag.

Returns
The definition entry if it exists

Return type
def_entry(DefinitionEntry or None)

DefValidator.**items**()

Return the dictionary of definitions.

Alias for .defs.items()

Returns
DefinitionEntry}): A list of definitions.

Return type
def_entries({str

DefValidator.**validate_def_tags**(*hed_string_obj*, *hed_validator=None*)

Validate Def/Def-Expand tags.

Parameters

- **hed_string_obj** (*HedString*) – The HED string to process.
- **hed_validator** (*HedValidator*) – Used to validate the placeholder replacement.

Returns
Issues found related to validating defs. Each issue is a dictionary.

Return type
list

DefValidator.**validate_def_value_units**(*def_tag*, *hed_validator*, *allow_placeholders=False*)

Equivalent to HedValidator.validate_units for the special case of a Def or Def-expand tag

DefValidator.**validate_onset_offset**(*hed_string_obj*)

Validate onset/offset

Parameters
hed_string_obj (*HedString*) – The HED string to check.

Returns
A list of issues found in validating onsets (i.e., out of order onsets, unknown def names).

Return type
list

DefValidator.**issues**

Return issues about duplicate definitions.

3.5.2 hed_validator

Top level validation of HED strings.

Classes

HedValidator(hed_schema[, def_dicts, ...])	Top level validation of HED strings.

3.5.2.1 HedValidator

class HedValidator(*hed_schema*, *def_dicts=None*, *definitions_allowed=False*)

Top level validation of HED strings.

This module contains the HedValidator class which is used to validate the tags in a HED string or a file. The file types include .tsv, .txt, and .xlsx. To get the validation issues after creating a HedValidator class call the get_validation_issues() function.

Methods

HedValidator.__init__(hed_schema[, ...])	Constructor for the HedValidator class.
HedValidator.check_tag_formatting(original_tag)	Report repeated or erroneous slashes.
HedValidator.run_basic_checks(hed_string, ...)	
HedValidator.run_full_string_checks(hed_string)	
HedValidator.validate(hed_string, ...[, ...])	Validate the string using the schema
HedValidator.validate_units(original_tag[, ...])	Validate units and value classes

Attributes

HedValidator.pattern_doubleslash

HedValidator.**__init__**(*hed_schema*, *def_dicts=None*, *definitions_allowed=False*)

Constructor for the HedValidator class.

Parameters

- **hed_schema** (*HedSchema or HedSchemaGroup*) – HedSchema object to use for validation.
- **def_dicts** (*DefinitionDict or list or dict*) – the def dicts to use for validation
- **definitions_allowed** (*bool*) – If False, flag definitions found as errors

HedValidator.**check_tag_formatting**(*original_tag*)

Report repeated or erroneous slashes.

Parameters

original_tag (*HedTag*) – The original tag that is used to report the error.

Returns

Validation issues. Each issue is a dictionary.

Return type

list

HedValidator.**run_basic_checks**(*hed_string*, *allow_placeholders*)

HedValidator.**run_full_string_checks**(*hed_string*)

HedValidator.**validate**(*hed_string*, *allow_placeholders*, *error_handler=None*)

Validate the string using the schema

Parameters

- **hed_string** (*HedString*) – the string to validate
- **allow_placeholders** (*bool*) – allow placeholders in the string
- **error_handler** (*ErrorHandler or None*) – the error handler to use, creates a default one if none passed

Returns

A list of issues for HED string

Return type

issues (list of dict)

HedValidator.**validate_units**(*original_tag*, *validate_text=None*, *report_as=None*, *error_code=None*, *index_offset=0*)

Validate units and value classes

Parameters

- **original_tag** (*HedTag*) – The source tag
- **validate_text** (*str*) – the text we want to validate, if not the full extension.
- **report_as** (*HedTag*) – Report the error tag as coming from a different one. Mostly for definitions that expand.

- **error_code** (*str*) – The code to override the error as. Again mostly for def/def-expand tags.
- **index_offset** (*int*) – Offset into the extension validate_text starts at

Returns
Issues found from units

Return type
issues(list)

HedValidator.**pattern_doubleslash = re.compile('([\\t/]{2,}|^/|/$)')**

3.5.3 onset_validator

Validates the onset/offset conditions.

Classes

OnsetValidator()	Validates onset/offset pairs.

3.5.3.1 OnsetValidator

class OnsetValidator
Validates onset/offset pairs.

Methods

OnsetValidator.__init__()	
OnsetValidator.check_for_banned_tags(hed_string)	Returns an issue for every tag found from the banned list (for files without onset column).
OnsetValidator.validate_temporal_relations(...)	Validate onset/offset/inset tag relations

Attributes

OnsetValidator.**__init__**()

static OnsetValidator.**check_for_banned_tags**(*hed_string*)
Returns an issue for every tag found from the banned list (for files without onset column).

Parameters
hed_string (*HedString*) – The string to check.

Returns
The validation issues associated with the characters. Each issue is dictionary.

Return type
list

OnsetValidator.**validate_temporal_relations**(*hed_string_obj*)

Validate onset/offset/inset tag relations

Parameters
hed_string_obj (*HedString*) – The HED string to check.

Returns
A list of issues found in validating onsets (i.e., out of order onsets, repeated def names).

Return type
list

3.5.4 reserved_checker

Classes

ReservedChecker()

3.5.4.1 ReservedChecker

class ReservedChecker

Methods

ReservedChecker.__init__()	
ReservedChecker.check_reserved_compatibility(...)	Check to make sure that the reserved tags can be used together and no duplicates.
ReservedChecker.check_tag_requirements(...)	Check the tag requirements within the group.
ReservedChecker.get_def_information(group, ...)	
ReservedChecker.get_group_requirements(...)	Returns the maximum and minimum number of groups required for these reserved tags.
ReservedChecker.get_incompatible(tag, ...)	Return the list of tags that cannot be in the same group with tag.
ReservedChecker.get_instance()	
ReservedChecker.get_reserved(group)	

Attributes

ReservedChecker.reserved_reqs_path

ReservedChecker.**__init__**()

ReservedChecker.**check_reserved_compatibility**(*group*, *reserved_tags*)

Check to make sure that the reserved tags can be used together and no duplicates.

Parameters

- **group** (*HedTagGroup*) – A group to be checked.
- **reserved_tags** (*list of HedTag*) – A list of reserved tags in this group.

ReservedChecker.**check_tag_requirements**(*group*, *reserved_tags*)

Check the tag requirements within the group.

Parameters

- **group** (*HedTagGroup*) – A group to be checked.
- **reserved_tags** (*list of HedTag*) – A list of reserved tags in this group.

Notes: This is only called when there are some reserved incompatible tags.

ReservedChecker.**get_def_information**(*group*, *reserved_tags*)

ReservedChecker.**get_group_requirements**(*reserved_tags*)

Returns the maximum and minimum number of groups required for these reserved tags.

Parameters
reserved_tags (*list of HedTag*) – The reserved tags to be checked.

Returns
tuple (max_required, min_required)

ReservedChecker.**get_incompatible**(*tag*, *reserved_tags*)

Return the list of tags that cannot be in the same group with tag.

Parameters

- **tag** (*HedTag*) –
- **reserved_tags** (*list of HedTag) - reserved tags (no duplicates*) –

Returns
list of HedTag

static ReservedChecker.**get_instance**()

ReservedChecker.**get_reserved**(*group*)

ReservedChecker.**reserved_reqs_path = '/home/docs/checkouts/readthedocs.org/user_builds/hed-python/checkouts/latest/hed/validator/data/reservedTags.json'**

3.5.5 sidecar_validator

Validates sidecars.

Classes

SidecarValidator(hed_schema)

3.5.5.1 SidecarValidator

class SidecarValidator(*hed_schema*)

Methods

SidecarValidator.__init__(hed_schema)	Constructor for the SidecarValidator class.
SidecarValidator.validate(sidecar[, ...])	Validate the input data using the schema
SidecarValidator.validate_structure(sidecar, ...)	Validate the raw structure of this sidecar.

Attributes

SidecarValidator.reserved_category_values
SidecarValidator.reserved_column_names

SidecarValidator.**__init__**(*hed_schema*)

Constructor for the SidecarValidator class.

Parameters
hed_schema (*HedSchema*) – HED schema object to use for validation.

SidecarValidator.**validate**(*sidecar*, *extra_def_dicts=None*, *name=None*, *error_handler=None*)

Validate the input data using the schema

Parameters

- **sidecar** (*Sidecar*) – Input data to be validated.
- **extra_def_dicts** (*list or DefinitionDict*) – extra def dicts in addition to sidecar
- **name** (*str*) – The name to report this sidecar as
- **error_handler** (*ErrorHandler*) – Error context to use. Creates a new one if None

Returns
A list of issues associated with each level in the HED string.

Return type
issues (list of dict)

SidecarValidator.**validate_structure**(*sidecar*, *error_handler*)

Validate the raw structure of this sidecar.

Parameters

- **sidecar** (*Sidecar*) – the sidecar to validate
- **error_handler** (*ErrorHandler*) – The error handler to use for error context

Returns
A list of issues found with the structure

Return type
issues(list)

SidecarValidator.**reserved_category_values = ['n/a']**

SidecarValidator.**reserved_column_names = ['HED']**

3.5.6 spreadsheet_validator

Validates spreadsheet tabular data.

Classes

SpreadsheetValidator(hed_schema)

3.5.6.1 SpreadsheetValidator

class SpreadsheetValidator(*hed_schema*)

Methods

SpreadsheetValidator.__init__(hed_schema)	Constructor for the SpreadsheetValidator class.
SpreadsheetValidator.validate(data[, ...])	Validate the input data using the schema

Attributes

SpreadsheetValidator.ONSET_TOLERANCE
SpreadsheetValidator.TEMPORAL_ANCHORS

SpreadsheetValidator.**__init__**(*hed_schema*)

Constructor for the SpreadsheetValidator class.

Parameters
hed_schema (*HedSchema*) – HED schema object to use for validation.

SpreadsheetValidator.**validate**(*data*, *def_dicts=None*, *name=None*, *error_handler=None*)

Validate the input data using the schema

Parameters

- **data** (*BaseInput*) – Input data to be validated.
- **def_dicts** (*list of DefDict or DefDict*) – all definitions to use for validation
- **name** (*str*) – The name to report errors from this file as
- **error_handler** (*ErrorHandler*) – Error context to use. Creates a new one if None

Returns

A list of issues for HED string

Return type

issues (list of dict)

SpreadsheetValidator.**ONSET_TOLERANCE = 3**

SpreadsheetValidator.**TEMPORAL_ANCHORS = re.compile('onset|inset|offset|delay')**

3.5.7 util

Validation of HED tags.

Modules

hed.validator.util.char_util	Classes responsible for basic character validation of a string or tag.
hed.validator.util.class_util	Utilities to support HED validation.
hed.validator.util.dup_util	
hed.validator.util.group_util	Validation of the HED tags as strings.
hed.validator.util.string_util	Utilities supporting the validation of HED strings.
hed.validator.util.tag_util	Utilities supporting validation of HED tags as strings.

3.5.7.1 char_util

Classes responsible for basic character validation of a string or tag.

Classes

CharRexValidator([modern_allowed_char_rules])	Class responsible for basic character level validation of a string or tag.
CharValidator([modern_allowed_char_rules])	Class responsible for basic character level validation of a string or tag.

3.5.7.1.1 CharRexValidator

class CharRexValidator(*modern_allowed_char_rules=False*)

Class responsible for basic character level validation of a string or tag.

Methods

CharRexValidator.__init__([...])	Does basic character validation for HED strings/tags
CharRexValidator.check_for_invalid_extension_chars(...)	Report invalid characters in extension/value.
CharRexValidator.check_invalid_character_issues(...)	Report invalid characters.
CharRexValidator.check_tag_invalid_chars(...)	Report invalid characters in the given tag.
CharRexValidator.get_problem_chars(in_str, cname)	
CharRexValidator.is_valid_value(in_string, cname)	

Attributes

CharRexValidator.DEFAULT_ALLOWED_PLACEHOLDER_CHARS
CharRexValidator.INVALID_STRING_CHARS
CharRexValidator.INVALID_STRING_CHARS_PLACEHOLDERS
CharRexValidator.TAG_ALLOWED_CHARS

CharRexValidator.**__init__**(*modern_allowed_char_rules=False*)

Does basic character validation for HED strings/tags

Parameters
modern_allowed_char_rules (*bool*) – If True, use 8.3 style rules for unicode characters.

CharRexValidator.**check_for_invalid_extension_chars**(*original_tag*, *validate_text*, *error_code=None*, *index_offset=0*)

Report invalid characters in extension/value.

Parameters

- **original_tag** (*HedTag*) – The original tag that is used to report the error.
- **validate_text** (*str*) – the text we want to validate, if not the full extension.
- **error_code** (*str*) – The code to override the error as. Again mostly for def/def-expand tags.
- **index_offset** (*int*) – Offset into the extension validate_text starts at.

Returns
Validation issues. Each issue is a dictionary.

Return type
list

CharRexValidator.**check_invalid_character_issues**(*hed_string*, *allow_placeholders*)

Report invalid characters.

Parameters

- **hed_string** (*str*) – A HED string.
- **allow_placeholders** (*bool*) – Allow placeholder and curly brace characters.

Returns
Validation issues. Each issue is a dictionary.

Return type
list

Notes

- **Invalid tag characters are defined by self.INVALID_STRING_CHARS or**
  self.INVALID_STRING_CHARS_PLACEHOLDERS

CharRexValidator.**check_tag_invalid_chars**(*original_tag*, *allow_placeholders*)

Report invalid characters in the given tag.

Parameters

- **original_tag** (*HedTag*) – The original tag that is used to report the error.
- **allow_placeholders** (*bool*) – Allow placeholder characters(#) if True.

Returns
Validation issues. Each issue is a dictionary.

Return type
list

CharRexValidator.**get_problem_chars**(*in_str*, *cname*)

CharRexValidator.**is_valid_value**(*in_string*, *cname*)

CharRexValidator.**DEFAULT_ALLOWED_PLACEHOLDER_CHARS = '.+-^ _#'**

CharRexValidator.**INVALID_STRING_CHARS = '[]{}~'**

CharRexValidator.**INVALID_STRING_CHARS_PLACEHOLDERS = '[]~'**

CharRexValidator.**TAG_ALLOWED_CHARS = '-_/'**

3.5.7.1.2 CharValidator

class CharValidator(*modern_allowed_char_rules=False*)

Class responsible for basic character level validation of a string or tag.

Methods

CharValidator.__init__([...])	Does basic character validation for HED strings/tags
CharValidator.check_for_invalid_extension_chars(...)	Report invalid characters in extension/value.
CharValidator.check_invalid_character_issues(...)	Report invalid characters.
CharValidator.check_tag_invalid_chars(...)	Report invalid characters in the given tag.

Attributes

CharValidator.DEFAULT_ALLOWED_PLACEHOLDER_CHARS
CharValidator.INVALID_STRING_CHARS
CharValidator.INVALID_STRING_CHARS_PLACEHOLDERS
CharValidator.TAG_ALLOWED_CHARS

CharValidator.**__init__**(*modern_allowed_char_rules=False*)

Does basic character validation for HED strings/tags

Parameters
modern_allowed_char_rules (*bool*) – If True, use 8.3 style rules for unicode characters.

CharValidator.**check_for_invalid_extension_chars**(*original_tag*, *validate_text*, *error_code=None*, *index_offset=0*)

Report invalid characters in extension/value.

Parameters

- **original_tag** (*HedTag*) – The original tag that is used to report the error.
- **validate_text** (*str*) – the text we want to validate, if not the full extension.
- **error_code** (*str*) – The code to override the error as. Again mostly for def/def-expand tags.
- **index_offset** (*int*) – Offset into the extension validate_text starts at.

Returns
Validation issues. Each issue is a dictionary.

Return type
list

CharValidator.**check_invalid_character_issues**(*hed_string*, *allow_placeholders*)

Report invalid characters.

Parameters

- **hed_string** (*str*) – A HED string.
- **allow_placeholders** (*bool*) – Allow placeholder and curly brace characters.

Returns
Validation issues. Each issue is a dictionary.

Return type
list

Notes

- **Invalid tag characters are defined by self.INVALID_STRING_CHARS or** self.INVALID_STRING_CHARS_PLACEHOLDERS

CharValidator.**check_tag_invalid_chars**(*original_tag*, *allow_placeholders*)

Report invalid characters in the given tag.

Parameters

- **original_tag** (*HedTag*) – The original tag that is used to report the error.
- **allow_placeholders** (*bool*) – Allow placeholder characters(#) if True.

Returns
Validation issues. Each issue is a dictionary.

Return type
list

CharValidator.**DEFAULT_ALLOWED_PLACEHOLDER_CHARS = '.+-^ _#'**

CharValidator.**INVALID_STRING_CHARS = '[]{}~'**

CharValidator.**INVALID_STRING_CHARS_PLACEHOLDERS = '[]~'**

CharValidator.**TAG_ALLOWED_CHARS = '-_/'**

3.5.7.2 class_util

Utilities to support HED validation.

Functions

find_invalid_positions(s, pattern)	
is_clock_face_time(time_string)	Check if a valid HH:MM time string.
is_date_time_value_class(date_time_string)	Check if the specified string is a valid datetime.
is_name_value_class(name_str)	
is_numeric_value_class(numeric_string)	Check to see if valid numeric value.
is_text_value_class(text_string)	Placeholder for eventual text value class validation.

find_invalid_positions(*s*, *pattern*)

is_clock_face_time(*time_string*)

Check if a valid HH:MM time string.

Parameters
time_string (*str*) – A time string.

Returns
True if the time string is valid. False, if otherwise.

Return type
bool

Notes

- This is deprecated and has no expected use going forward.

`is_date_time_value_class`(*date_time_string*)

Check if the specified string is a valid datetime.

Parameters
date_time_string (*str*) – A datetime string.

Returns
True if the datetime string is valid. False, if otherwise.

Return type
bool

Notes

- ISO 8601 datetime string.

`is_name_value_class`(*name_str*)

`is_numeric_value_class`(*numeric_string*)

Check to see if valid numeric value.

Parameters
numeric_string (*str*) – A string that should be only a number with no units.

Returns
True if the numeric string is valid. False, if otherwise.

Return type
bool

`is_text_value_class`(*text_string*)

Placeholder for eventual text value class validation.

Parameters
text_string (*str*) – Text class.

Returns
True

Return type
bool

Classes

UnitValueValidator([...])	Validates units.

3.5.7.2.1 UnitValueValidator

class UnitValueValidator(*modern_allowed_char_rules=False*, *value_validators=None*)

Validates units.

Methods

UnitValueValidator.__init__([...])	Validates the unit and value classes on a given tag.
UnitValueValidator.check_tag_unit_class_units_are_valid(...)	Report incorrect unit class or units.
UnitValueValidator.check_tag_value_class_valid(...)	Report an invalid value portion.
UnitValueValidator.report_value_char_errors(...)	
UnitValueValidator.report_value_errors(...)	
UnitValueValidator.validate_value_class_type(...)	Report invalid unit or valid class values.

Attributes

UnitValueValidator.DATE_TIME_VALUE_CLASS
UnitValueValidator.DIGIT_OR_POUND_EXPRESSION
UnitValueValidator.NAME_VALUE_CLASS
UnitValueValidator.NUMERIC_VALUE_CLASS
UnitValueValidator.TEXT_VALUE_CLASS

UnitValueValidator.**__init__**(*modern_allowed_char_rules=False*, *value_validators=None*)

Validates the unit and value classes on a given tag.

Parameters

value_validators (*dict or None*) – Override or add value class validators

UnitValueValidator.**check_tag_unit_class_units_are_valid**(*original_tag*, *validate_text*, *report_as=None*, *error_code=None*)

Report incorrect unit class or units.

Parameters

- **original_tag** (*HedTag*) – The original tag that is used to report the error.

- **validate_text** (*str*) – The text to validate.
- **report_as** (*HedTag*) – Report errors as coming from this tag, rather than original_tag.
- **error_code** (*str*) – Override error codes.

Returns
Validation issues. Each issue is a dictionary.

Return type
list

UnitValueValidator.**check_tag_value_class_valid**(*original_tag*, *validate_text*, *report_as=None*)

Report an invalid value portion.

Parameters

- **original_tag** (*HedTag*) – The original tag that is used to report the error.
- **validate_text** (*str*) – The text to validate.
- **report_as** (*HedTag*) – Report errors as coming from this tag, rather than original_tag.

Returns
Validation issues.

Return type
list

static UnitValueValidator.**report_value_char_errors**(*class_name*, *errors*, *report_as*)

static UnitValueValidator.**report_value_errors**(*error_dict*, *class_valid*, *report_as*)

UnitValueValidator.**validate_value_class_type**(*unit_or_value_portion*, *valid_types*)

Report invalid unit or valid class values.

Parameters

- **unit_or_value_portion** (*str*) – The value portion to validate.
- **valid_types** (*list*) – The names of value class or unit class types (e.g. dateTime or dateTimeClass).

Returns
True if this is one of the valid_types validators.

Return type
type_valid (bool)

UnitValueValidator.**DATE_TIME_VALUE_CLASS = 'dateTimeClass'**

UnitValueValidator.**DIGIT_OR_POUND_EXPRESSION = '^(-?[\\d.]+(?:e-?\\d+)?|#)$'**

UnitValueValidator.**NAME_VALUE_CLASS = 'nameClass'**

UnitValueValidator.**NUMERIC_VALUE_CLASS = 'numericClass'**

UnitValueValidator.**TEXT_VALUE_CLASS = 'textClass'**

3.5.7.3 dup_util

Classes

DuplicateChecker()

3.5.7.3.1 DuplicateChecker

class DuplicateChecker

Methods

DuplicateChecker.__init__()	Checker for duplications in HED groups.
DuplicateChecker.check_for_duplicates(group)	Find duplicates in a HED group and return the errors found.
DuplicateChecker.get_hash(group)	Return the unique hash for the group as long as no duplicates.

Attributes

DuplicateChecker.**__init__**()

Checker for duplications in HED groups.

Notes

This checker has an early out strategy – it returns when it finds an error.

DuplicateChecker.**check_for_duplicates**(*group*)

Find duplicates in a HED group and return the errors found.

Parameters
group (*HedGroup*) – The HED group to be checked.

Returns
List of validation issues – which might be empty if no duplicates detected.

Return type
list

DuplicateChecker.**get_hash**(*group*)

Return the unique hash for the group as long as no duplicates.

Parameters
group (*HedGroup*) – The HED group to be checked.

Returns
Unique hash or None if duplicates were detected within the group.

Return type
int or None

3.5.7.4 group_util

Validation of the HED tags as strings.

Classes

GroupValidator(hed_schema)	Validation for attributes across groups HED tags.

3.5.7.4.1 GroupValidator

class GroupValidator(*hed_schema*)

Validation for attributes across groups HED tags.

This is things like Required, Unique, top level tags, etc.

Methods

GroupValidator.__init__(hed_schema)	Constructor for GroupValidator
GroupValidator.check_for_required_tags(tags)	Report missing required tags.
GroupValidator.check_multiple_unique_tags_exist(tags)	Report if multiple identical unique tags exist
GroupValidator.check_tag_level_issue(...)	Report tags incorrectly positioned in hierarchy.
GroupValidator.run_all_tags_validators(...)	Report invalid the multi-tag properties in a HED string, e.g.
GroupValidator.run_tag_level_validators(...)	Report invalid groups at each level.
GroupValidator.validate_duration_tags(...)	Validate Duration/Delay tag groups

Attributes

GroupValidator.**__init__**(*hed_schema*)

Constructor for GroupValidator

Parameters
hed_schema (*HedSchema*) – A HedSchema object.

GroupValidator.**check_for_required_tags**(*tags*)

Report missing required tags.

Parameters
tags (*list*) – HedTags containing the tags.

Returns
Validation issues. Each issue is a dictionary.

Return type
list

GroupValidator.**check_multiple_unique_tags_exist**(*tags*)

Report if multiple identical unique tags exist

A unique Term can only appear once in a given HedString. Unique terms are terms with the 'unique' property in the schema.

Parameters
tags (*list*) – HedTags containing the tags.

Returns
Validation issues. Each issue is a dictionary.

Return type
list

static GroupValidator.**check_tag_level_issue**(*original_tag_list*, *is_top_level*, *is_group*)

Report tags incorrectly positioned in hierarchy.

Parameters

- **original_tag_list** (*list of HedTag*) – HedTags containing the original tags.
- **is_top_level** (*bool*) – If True, this group is a "top level tag group".
- **is_group** (*bool*) – If True group should be contained by parenthesis.

Returns
Validation issues. Each issue is a dictionary.

Return type
list

GroupValidator.**run_all_tags_validators**(*hed_string_obj*)

Report invalid the multi-tag properties in a HED string, e.g. required tags.

Parameters
hed_string_obj (*HedString*) – A HedString object.

Returns
The issues associated with the tags in the HED string. Each issue is a dictionary.

Return type
list

GroupValidator.**run_tag_level_validators**(*hed_string_obj*)

Report invalid groups at each level.

Parameters
hed_string_obj (*HedString*) – A HedString object.

Returns
Issues associated with each level in the HED string. Each issue is a dictionary.

Return type
list

Notes

- This pertains to the top-level, all groups, and nested groups.

static GroupValidator.**validate_duration_tags**(*hed_string_obj*)

Validate Duration/Delay tag groups

Parameters
hed_string_obj (*HedString*) – The HED string to check.

Returns
Issues found in validating durations (i.e., extra tags or groups present, or a group missing)

Return type
list

3.5.7.5 string_util

Utilities supporting the validation of HED strings.

Classes

StringValidator()	Runs checks on the raw string that depend on multiple characters, e.g.

3.5.7.5.1 StringValidator

class StringValidator

Runs checks on the raw string that depend on multiple characters, e.g. mismatched parentheses

Methods

StringValidator.__init__()	
StringValidator.check_count_tag_group_parentheses(...)	Report unmatched parentheses.
StringValidator.check_delimiter_issues_in_hed_string(...)	Report missing commas or commas in value tags.
StringValidator.run_string_validator(...)	

Attributes

StringValidator.CLOSING_GROUP_CHARACTER	
StringValidator.COMMA	
StringValidator.OPENING_GROUP_CHARACTER	

StringValidator.__init__()

static StringValidator.check_count_tag_group_parentheses(*hed_string*)

Report unmatched parentheses.

Parameters
hed_string (*str*) – A HED string.

Returns
A list of validation list. Each issue is a dictionary.

Return type
list

StringValidator.check_delimiter_issues_in_hed_string(*hed_string*)

Report missing commas or commas in value tags.

Parameters
hed_string (*str*) – A HED string.

Returns
A validation issues list. Each issue is a dictionary.

Return type
list

StringValidator.run_string_validator(*hed_string_obj*)

StringValidator.CLOSING_GROUP_CHARACTER = ')'

StringValidator.COMMA = ','

StringValidator.OPENING_GROUP_CHARACTER = '('

3.5.7.6 tag_util

Utilities supporting validation of HED tags as strings.

Classes

TagValidator()	Validation for individual HED tags.

3.5.7.6.1 TagValidator

class TagValidator

Validation for individual HED tags.

Methods

TagValidator.__init__()	
TagValidator.check_capitalization(original_tag)	Report warning if incorrect tag capitalization.
TagValidator.check_for_placeholder(original_tag)	Report invalid placeholder characters.
TagValidator.check_tag_exists_in_schema(...)	Report invalid tag or doesn't take a value.
TagValidator.check_tag_is_deprecated(...)	
TagValidator.check_tag_requires_child(...)	Report if tag is a leaf with 'requiredTag' attribute.
TagValidator.run_individual_tag_validators(...)	Runs the validators on the individual tags.

Attributes

TagValidator.CAMEL_CASE_EXPRESSION

TagValidator.**__init__**()

TagValidator.**check_capitalization**(*original_tag*)

Report warning if incorrect tag capitalization.

Parameters
original_tag (*HedTag*) – The original tag used to report the warning.

Returns
Validation issues. Each issue is a dictionary.

Return type
list

static TagValidator.**check_for_placeholder**(*original_tag*, *is_definition=False*)

Report invalid placeholder characters.

Parameters

- **original_tag** (*HedTag*) – The HedTag to be checked
- **is_definition** (*bool*) – If True, placeholders are allowed.

Returns
Validation issues. Each issue is a dictionary.

Return type
list

Notes

- Invalid placeholder may appear in the extension/value portion of a tag.

static TagValidator.**check_tag_exists_in_schema**(*original_tag*)

Report invalid tag or doesn't take a value.

Parameters
original_tag (*HedTag*) – The original tag that is used to report the error.

Returns
Validation issues. Each issue is a dictionary.

Return type
list

TagValidator.**check_tag_is_deprecated**(*original_tag*)

static TagValidator.**check_tag_requires_child**(*original_tag*)

Report if tag is a leaf with 'requiredTag' attribute.

Parameters
original_tag (*HedTag*) – The original tag that is used to report the error.

Returns
Validation issues. Each issue is a dictionary.

Return type
list

TagValidator.**run_individual_tag_validators**(*original_tag*, *allow_placeholders=False*, *is_definition=False*)

Runs the validators on the individual tags.

This ignores most illegal characters except in extensions.

Parameters

- **original_tag** (*HedTag*) – A original tag.
- **allow_placeholders** (*bool*) – Allow value class or extensions to be placeholders rather than a specific value.
- **is_definition** (*bool*) – This tag is part of a Definition, not a normal line.

Returns
The validation issues associated with the tags. Each issue is dictionary.

Return type
list

TagValidator.**CAMEL_CASE_EXPRESSION = '([A-Z]+\\s*[a-z-]*)+'**

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

h

INDEX

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

X