IEML Library Documentation

Release 0.1.0

Hadrien Titeux

Mar 08, 2021

Table of Content

This is the home of the documentation for the Python implementation of the IEML metalanguage. You should find here

- a brief intro to the theory behind the IEML metalanguage
- examples on how to use our implementation
- an API reference

Bear in mind that the project is still under heavy development (and conception). You should use this documentation as a way to get a better understanding of the IEML project, and a basic overview of our Python implementation's usage.

CHAPTER 1

An Overview of the IEML metalanguage

CHAPTER 2

An introduction to the IEML Library

CHAPTER 3

API Reference

3.1 Dictionary reference

3.1.1 Script Structure

class ieml.dictionary.script.script.**AdditiveScript**(*children=None*, *character=None*)
Bases: *ieml.dictionary.script.script.Script*

Represent an addition of same layer scripts.

__init__(*children=None*, *character=None*)
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'ieml.dictionary.script.script'

class ieml.dictionary.script.script.**MultiplicativeScript**(*substance=None*, *attribute=None*, *mode=None*, *children=None*, *character=None*)
Bases: *ieml.dictionary.script.script.Script*

Represent a multiplication of three scripts of the same layer.

__init__(*substance=None*, *attribute=None*, *mode=None*, *children=None*, *character=None*)
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'ieml.dictionary.script.script'

class ieml.dictionary.script.script.**NullScript**(*layer*)
Bases: *ieml.dictionary.script.script.Script*

__init__(*layer*)
Initialize self. See help(type(self)) for accurate signature.

__iter__()
Enables the syntactic sugar of iterating directly on an element without accessing "children"

__module__ = 'ieml.dictionary.script.script'

class ieml.dictionary.script.script.**Script**(*children=None*, *character=None*, **args*, ***kwargs*)

Bases: *ieml.commons.TreeStructure*, *ieml.commons.DecoratedComponent*

A parser is defined by a character (PRIMITIVES, REMARKABLE_ADDITION OR REMARKABLE_MULTIPLICATION) or a list of parser children. All the element in the children list must be an AdditiveScript or a MultiplicativeScript.

__add__(*other*)

__contains__(*item*)

__eq__(*other*)
Return self==value.

__getnewargs_ex__()

__hash__()
Since the IEML string for a script is its definition, it can be used as a hash

__init__(*children=None*, *character=None*, **args*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

__len__()

__lt__(*other*)
Return self<value.

__module__ = 'ieml.dictionary.script.script'

static __new__(*cls*, **args*, ***kwargs*)
Need this to pickle scripts, the pickler use __hash__ method before unpickling the object attribute. Then need to pass the _str.

cells

check()

headers

is_singular

iter_structure()

iter_structure_path(*flexion=False*)

iter_structure_path_by_script_ss(*flexion=False*)

singular_sequences

singular_sequences_set

tables_script

ieml.dictionary.script.operator.**add**(*scripts: List[ieml.dictionary.script.script.Script]*)

ieml.dictionary.script.operator.**m**(*substance*, *attribute=None*, *mode=None*)

ieml.dictionary.script.operator.**script**(*arg*, *promote=False*, *factorize=False*)

ieml.dictionary.script.tools.**factor**(*sequences*)

ieml.dictionary.script.tools.**factorize**(*script: Union[ieml.dictionary.script.script.Script, List[ieml.dictionary.script.script.Script]]*, *promote: bool = True*) → ieml.dictionary.script.script.Script

Parameters

- **script** – The Script or list of Script to factorize
- **promote** – If script is a list, promote all Script to the layer max(sc.layer for sc in scripts)

Returns the factorized script

ieml.dictionary.script.tools.**pack_factorisation**(*facto_list*)

Parameters **facto_list** – list of script or tuple of factorisation

Returns

ieml.dictionary.script.tools.**promote**(*script: ieml.dictionary.script.script.Script*, *layer: int*)
Promote script to layer by multiplying it with null scripts (E:) :param script: :param layer: :return:

3.1.2 Dictionary Structure

class ieml.dictionary.dictionary.**Dictionary**(*paradigms*, *structure*)
Bases: object

__contains__(*item*)

__dict__ = mappingproxy({'__module__': 'ieml.dictionary.dictionary', '__init__': <fu

__getitem__(*item*)

__init__(*paradigms*, *structure*)
Initialize self. See help(type(self)) for accurate signature.

__len__()

__module__ = 'ieml.dictionary.dictionary'

__weakref__
list of weak references to the object (if defined)

3.1.3 Script Parser

ieml.dictionary.script.parser.lexer.**get_script_lexer**(*module=None*)

class ieml.dictionary.script.parser.parser.**ScriptParser**
Bases: object

__dict__ = mappingproxy({'__module__': 'ieml.dictionary.script.parser.parser', 'token

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'ieml.dictionary.script.parser.parser'

__weakref__
list of weak references to the object (if defined)

lock = <unlocked _thread.lock object>

p_additive_script_lvl_0(*p*)
additive_script_lvl_0 : sum_lvl_0

p_additive_script_lvl_1(*p*)
additive_script_lvl_1 : sum_lvl_1

p_additive_script_lvl_2 (*p*)
additive_script_lvl_2 : sum_lvl_2

p_additive_script_lvl_3 (*p*)
additive_script_lvl_3 : sum_lvl_3

p_additive_script_lvl_4 (*p*)
additive_script_lvl_4 : sum_lvl_4

p_additive_script_lvl_5 (*p*)
additive_script_lvl_5 : sum_lvl_5

p_additive_script_lvl_6 (*p*)
additive_script_lvl_6 : sum_lvl_6

p_error (*p*)

p_script_lvl_0 (*p*)
script_lvl_0 : PRIMITIVE LAYER0_MARK | REMARKABLE_ADDITION LAYER0_MARK

p_script_lvl_1 (*p*)
script_lvl_1 : additive_script_lvl_0 LAYER1_MARK | additive_script_lvl_0 additive_script_lvl_0 LAYER1_MARK | additive_script_lvl_0 additive_script_lvl_0 additive_script_lvl_0 LAYER1_MARK | REMARKABLE_MULTIPLICATION LAYER1_MARK

p_script_lvl_2 (*p*)
script_lvl_2 : sum_lvl_1 LAYER2_MARK | sum_lvl_1 sum_lvl_1 LAYER2_MARK | sum_lvl_1 sum_lvl_1 sum_lvl_1 LAYER2_MARK

p_script_lvl_3 (*p*)
script_lvl_3 : sum_lvl_2 LAYER3_MARK | sum_lvl_2 sum_lvl_2 LAYER3_MARK | sum_lvl_2 sum_lvl_2 sum_lvl_2 LAYER3_MARK

p_script_lvl_4 (*p*)
script_lvl_4 : sum_lvl_3 LAYER4_MARK | sum_lvl_3 sum_lvl_3 LAYER4_MARK | sum_lvl_3 sum_lvl_3 sum_lvl_3 LAYER4_MARK

p_script_lvl_5 (*p*)
script_lvl_5 : sum_lvl_4 LAYER5_MARK | sum_lvl_4 sum_lvl_4 LAYER5_MARK | sum_lvl_4 sum_lvl_4 sum_lvl_4 LAYER5_MARK

p_script_lvl_6 (*p*)
script_lvl_6 : sum_lvl_5 LAYER6_MARK | sum_lvl_5 sum_lvl_5 LAYER6_MARK | sum_lvl_5 sum_lvl_5 sum_lvl_5 LAYER6_MARK

p_sum_lvl_0 (*p*)
sum_lvl_0 : script_lvl_0 | script_lvl_0 PLUS sum_lvl_0

p_sum_lvl_1 (*p*)
sum_lvl_1 : script_lvl_1 | script_lvl_1 PLUS sum_lvl_1

p_sum_lvl_2 (*p*)
sum_lvl_2 : script_lvl_2 | script_lvl_2 PLUS sum_lvl_2

p_sum_lvl_3 (*p*)
sum_lvl_3 : script_lvl_3 | script_lvl_3 PLUS sum_lvl_3

p_sum_lvl_4 (*p*)
sum_lvl_4 : script_lvl_4 | script_lvl_4 PLUS sum_lvl_4

p_sum_lvl_5 (*p*)
sum_lvl_5 : script_lvl_5 | script_lvl_5 PLUS sum_lvl_5

p_sum_lvl_6(*p*)
sum_lvl_6 : script_lvl_6 | script_lvl_6 PLUS sum_lvl_6

p_term(*p*)
term : script_lvl_0 | additive_script_lvl_0 | script_lvl_1 | additive_script_lvl_1 | script_lvl_2 | additive_script_lvl_2 | script_lvl_3 | additive_script_lvl_3 | script_lvl_4 | additive_script_lvl_4 | script_lvl_5 | additive_script_lvl_5 | script_lvl_6 | additive_script_lvl_6

t_add_rules()

t_parse

tokens = ('PLUS', 'LAYER0_MARK', 'LAYER1_MARK', 'LAYER2_MARK', 'LAYER3_MARK', 'LAYER4_

3.2 USL Object Reference

ieml.usl.**get_index**(*s*, *dic*)

ieml.usl.**int2base**(*i*, *max=30*, *characters=['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']*)

3.2.1 USL Object Structure

class ieml.usl.usl.**USL**(**args*, ***kwargs*)
Bases: *ieml.commons.DecoratedComponent*

__bool__()

__contains__(*item*)

__eq__(*other*)
Return self==value.

__hash__()
Since the IEML string for a script is its definition, it can be used as a hash

__init__(**args*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

__len__()

__lt__(*other*)
Return self<value.

__module__ = 'ieml.usl.usl'

__str__()
Return str(self).

cardinal

check()

do_lt(*other*)

empty

is_singular

iter_structure()

iter_structure_path(*flexion=False*) → Iterable[Tuple[UslPath, USL]]

iter_structure_path_by_script_ss(*flexion=False*) → Iterable[Tuple[UslPath, ieml.dictionary.script.script.Script]]

iter_structure_path_by_type(*_type=<class 'ieml.dictionary.script.script.Script'>*, *flexion=False*)

morphemes

singular_sequences

singular_sequences_set

syntactic_level = 0

ieml.usl.usl.**usl**(*arg: Union[str, ieml.dictionary.script.script.Script, ieml.usl.usl.USL, Iterable[Tuple[UslPath, Union[ieml.usl.usl.USL, ieml.dictionary.script.script.Script]]]]*) → ieml.usl.usl.USL

Cast argument to an USL type, depending on the argument type.

- If argument is a string, it is parsed by ieml.usl.parser.IEMLParser.parse
- if argument is a ieml.dictionary.Script, the returned object is a ieml.usl.polymorpheme.PolyMorpheme with the argument as the constant.
- if argument is an ieml.usl.usl.USL, the argument is returned
- if argument is a list of (ieml.usl.decoration.path.UslPath, ieml.usl.usl.USL)

Parameters **arg** (*Union[str, Script, USL, Iterable[Tuple['UslPath', Union[USL, Script]]]]*) –

Returns an ieml.usl.usl.USL

class ieml.usl.polymorpheme.**PolyMorpheme**(*constant: List[ieml.dictionary.script.script.Script] = ()*, *groups=()*)

Bases: *ieml.usl.usl.USL*

__init__(*constant: List[ieml.dictionary.script.script.Script] = ()*, *groups=()*)
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'ieml.usl.polymorpheme'

check()

do_lt(*other*)

empty

iter_structure()

iter_structure_path(*flexion=False*)

morphemes

syntactic_level = 1

ieml.usl.polymorpheme.**check_polymorpheme**(*ms*)

ieml.usl.polymorpheme.**compute_PM_singular_sequences**(*constants*, *groups*)

class ieml.usl.lexeme.**Lexeme**(*pm_flexion: ieml.usl.polymorpheme.PolyMorpheme*, *pm_content: ieml.usl.polymorpheme.PolyMorpheme*)

Bases: *ieml.usl.usl.USL*

A lexeme without the PA of the position on the tree (position independant lexeme)

__init__(*pm_flexion: ieml.usl.polymorpheme.PolyMorpheme*, *pm_content: ieml.usl.polymorpheme.PolyMorpheme*)

Initialize self. See help(type(self)) for accurate signature.

__module__ = 'ieml.usl.lexeme'

check()

do_lt(*other*)

empty

iter_structure()

iter_structure_path(*flexion=False*)

morphemes

syntactic_level = 2

ieml.usl.lexeme.**check_lexeme**(*lexeme*, *sfun=None*)

class ieml.usl.syntagmatic_function.**DependantQualitySyntagmaticFunction**(*actor: Any*, *dependant: Union[DependantQualitySy ieml.usl.syntagmatic_functi = None*, *independant: Union[ieml.usl.syntagmatic ieml.usl.syntagmatic_functi = None*, ***kwargs*)

Bases: *ieml.usl.syntagmatic_function.SyntagmaticFunction*

__init__(*actor: Any*, *dependant: Union[DependantQualitySyntagmaticFunction, ieml.usl.syntagmatic_function.JunctionSyntagmaticFunction] = None*, *independant: Union[ieml.usl.syntagmatic_function.IndependantQualitySyntagmaticFunction, ieml.usl.syntagmatic_function.JunctionSyntagmaticFunction] = None*, ***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

__module__ = 'ieml.usl.syntagmatic_function'

check(*X*, *check_X*, *sfun_type*)

class ieml.usl.syntagmatic_function.**IndependantQualitySyntagmaticFunction**(*actor: Any*)

Bases: *ieml.usl.syntagmatic_function.SyntagmaticFunction*

__init__(*actor: Any*)
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'ieml.usl.syntagmatic_function'

class ieml.usl.syntagmatic_function.**JunctionSyntagmaticFunction**(*junction_link: ieml.dictionary.script.script.Script, children: List[ieml.usl.syntagmatic_function.Synt*

Bases: *ieml.usl.syntagmatic_function.SyntagmaticFunction*

__init__(*junction_link: ieml.dictionary.script.script.Script, children: List[ieml.usl.syntagmatic_function.SyntagmaticFunction]*)
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'ieml.usl.syntagmatic_function'

check(*X: Type[CT_co], check_X, sfun_type*)

class ieml.usl.syntagmatic_function.**ProcessSyntagmaticFunction**(*actor: Any, valence: ieml.dictionary.script.script.Script, actants: Dict[ieml.dictionary.script.script.Script, ieml.usl.syntagmatic_function.Dependan*

Bases: *ieml.usl.syntagmatic_function.SyntagmaticFunction*

__init__(*actor: Any, valence: ieml.dictionary.script.script.Script, actants: Dict[ieml.dictionary.script.script.Script, ieml.usl.syntagmatic_function.DependantQualitySyntagmaticFunction]*)
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'ieml.usl.syntagmatic_function'

check(*X: Type[CT_co], check_X, sfun_type*)

class ieml.usl.syntagmatic_function.**SyntagmaticFunction**(*actor: Any, _actors: Dict[List[ieml.dictionary.script.script.Script], SyntagmaticFunction]*)

Bases: object

__dict__ = mappingproxy({'__module__': 'ieml.usl.syntagmatic_function', '__init__':

__eq__(*other*)
Return self==value.

__hash__ = None

__init__(*actor: Any, _actors: Dict[List[ieml.dictionary.script.script.Script], SyntagmaticFunction]*)
Initialize self. See help(type(self)) for accurate signature.

__lt__(*other*)
Return self<value.

__module__ = 'ieml.usl.syntagmatic_function'

__weakref__
list of weak references to the object (if defined)

as_list(*context_type*) → List[Tuple[List[ieml.dictionary.script.script.Script], Any]]

check(*X: Type[CT_co], check_X, sfun_type*)

empty

static from_list(*l: List[Tuple[List[ieml.dictionary.script.script.Script], Any]]*) → Tuple[Type[ieml.usl.syntagmatic_function.SyntagmaticFunction], ieml.usl.syntagmatic_function.SyntagmaticFunction]

get(*role: ieml.usl.syntagmatic_function.SyntagmaticRole*, *ignore_prefix=False*, *ignore_process_valence=False*) → Any

static get_context_role_prefix(*context*)

get_paradigm(*role: ieml.usl.syntagmatic_function.SyntagmaticRole*) → List[Any]

get_role_expansion(*role: ieml.usl.syntagmatic_function.SyntagmaticRole*, *ignore_prefix=()*)

iter_structure()

iter_structure_path(*context*, *focus_role=None*)

render_with_context(*role: ieml.usl.syntagmatic_function.SyntagmaticRole = None*, *context=None*)

role_in(*tgt: ieml.usl.syntagmatic_function.SyntagmaticFunction*)
return the role of the tgt syntagmatic function occupy in self.

role_is_junction(*role: ieml.usl.syntagmatic_function.SyntagmaticRole = None*)

singular_sequences(*context_type*)

class ieml.usl.syntagmatic_function.**SyntagmaticRole**(*constant: List[ieml.dictionary.script.script.Script] = ()*)

Bases: object

__dict__ = mappingproxy({'__module__': 'ieml.usl.syntagmatic_function', '__init__':

__eq__(*other*)
Return self==value.

__hash__()
Since the IEML string for a script is its definition, it can be used as a hash

__init__(*constant: List[ieml.dictionary.script.script.Script] = ()*)
Initialize self. See help(type(self)) for accurate signature.

__lt__(*other*)
Return self<value.

__module__ = 'ieml.usl.syntagmatic_function'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

is_junction_prefix(*role: ieml.usl.syntagmatic_function.SyntagmaticRole*)
if self is the junction prefix of the role

class ieml.usl.word.**Word**(*syntagmatic_fun: ieml.usl.syntagmatic_function.SyntagmaticFunction, role: ieml.usl.syntagmatic_function.SyntagmaticRole, context_type: Type[ieml.usl.syntagmatic_function.SyntagmaticFunction]*)

Bases: *ieml.usl.usl.USL*

__init__(*syntagmatic_fun: ieml.usl.syntagmatic_function.SyntagmaticFunction, role: ieml.usl.syntagmatic_function.SyntagmaticRole, context_type: Type[ieml.usl.syntagmatic_function.SyntagmaticFunction]*)
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'ieml.usl.word'

check()

do_lt(*other*)

empty

iter_structure()

iter_structure_path(*flexion=False*)

morphemes

syntactic_level = 3

ieml.usl.word.**check_word**(*w: ieml.usl.word.Word*)

ieml.usl.word.**simplify_word**(*w: ieml.usl.word.Word*) → ieml.usl.word.Word
remove empty leaves

3.2.2 Table Structure

class ieml.usl.table.**UslTable2D**(*usl: ieml.usl.usl.USL*, *columns: ieml.usl.decoration.path.UslPath*, *rows: ieml.usl.decoration.path.UslPath = None*)
Bases: object

__dict__ = mappingproxy({'__module__': 'ieml.usl.table', '__init__': <function UslTa

__init__(*usl: ieml.usl.usl.USL*, *columns: ieml.usl.decoration.path.UslPath*, *rows: ieml.usl.decoration.path.UslPath = None*)
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'ieml.usl.table'

__weakref__
list of weak references to the object (if defined)

cells

column_paths_constant

column_paths_variation

columns

constant_paths

row_paths_constant

row_paths_variation

rows

ieml.usl.table.**enumerate_partitions**(*usl: ieml.usl.usl.USL*)

class ieml.usl.variation.**PolyMorphemeVariation**(*items*, *multiplicity*)
Bases: *ieml.usl.variation.Variation*

__init__(*items*, *multiplicity*)
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'ieml.usl.variation'

check()

do_lt(*other*)

empty

iter_structure()

iter_structure_path(*flexion=False*)

morphemes

class ieml.usl.variation.**Variation**(**args*, ***kwargs*)
Bases: *ieml.usl.usl.USL*

__module__ = 'ieml.usl.variation'

3.2.3 USL Parser

ieml.usl.parser.lexer.**get_lexer**(*module=None*)

class ieml.usl.parser.parser.**IEMLParser**(*dictionary=None*)
Bases: object

__dict__ = mappingproxy({'__module__': 'ieml.usl.parser.parser', 'tokens': ('MORPHEM

__init__(*dictionary=None*)
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'ieml.usl.parser.parser'

__weakref__
list of weak references to the object (if defined)

lock = <unlocked _thread.lock object>

p_decoration(*p*)
decoration : LBRACKET USL_PATH DECORATION_VALUE RBRACKET

p_decoration_list(*p*)
decoration_list : decoration_list decoration | decoration

p_error(*p*)

p_group(*p*)
group : GROUP_MULTIPLICITY LPAREN morpheme_sum RPAREN

p_group_list(*p*)
group_list : group group_list | group

p_ieml_proposition(*p*)
proposition : morpheme | usl | instanced_usl

p_instanced_usl(*p*)
instanced_usl : usl decoration_list

p_lexeme(*p*)
lexeme : LPAREN poly_morpheme RPAREN LPAREN poly_morpheme RPAREN LPAREN poly_morpheme RPAREN | LPAREN RPAREN LPAREN poly_morpheme RPAREN LPAREN poly_morpheme RPAREN | LPAREN RPAREN LPAREN RPAREN LPAREN poly_morpheme RPAREN | LPAREN poly_morpheme RPAREN LPAREN poly_morpheme RPAREN | LPAREN RPAREN LPAREN poly_morpheme RPAREN | LPAREN poly_morpheme RPAREN | LPAREN RPAREN

p_lexeme_list(*p*)
lexeme_list : lexeme_list RCHEVRON EXCLAMATION_MARK positioned_lexeme | lexeme_list RCHEVRON positioned_lexeme | EXCLAMATION_MARK positioned_lexeme | positioned_lexeme

p_morpheme(*p*)
morpheme : MORPHEME

p_morpheme_sum(*p*)
morpheme_sum : morpheme_sum morpheme | morpheme

p_poly_morpheme(*p*)
poly_morpheme : morpheme_sum group_list | morpheme_sum | group_list

p_positioned_lexeme(*p*)
positioned_lexeme : morpheme_sum lexeme | lexeme

p_usl(*p*)
usl : poly_morpheme | lexeme | word

p_word(*p*)
word : LBRACKET OLD_MORPHEME_GRAMMATICAL_CLASS lexeme_list RBRACKET | LBRACKET lexeme_list RBRACKET

parse(*s*, *factorize_script=False*)
Parses the input string, and returns a reference to the created AST's root

tokens = ('MORPHEME', 'OLD_MORPHEME_GRAMMATICAL_CLASS', 'LPAREN', 'RPAREN', 'RCHEVRON'

class ieml.usl.parser.parser.**IEMLParserSingleton**
Bases: type

__call__(**args*, ***kwargs*)
Call self as a function.

__module__ = 'ieml.usl.parser.parser'

3.2.4 Path Parser

exception ieml.usl.decoration.path.**DeferenceError**
Bases: KeyError

__module__ = 'ieml.usl.decoration.path'

__weakref__
list of weak references to the object (if defined)

class ieml.usl.decoration.path.**FlexionPath**(*morpheme*, *child=None*)
Bases: *ieml.usl.decoration.path.UslPath*

USL_TYPE
alias of *ieml.usl.polymorpheme.PolyMorpheme*

__init__(*morpheme*, *child=None*)
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'ieml.usl.decoration.path'

as_constant(*u=None*)

classmethod build_usl_from_path_to_node(*path_to_node*)

clone(*use_child=False*, *child=None*)

without_morpheme()

class ieml.usl.decoration.path.**GroupIndex**
Bases: *ieml.commons.OrderedEnum*

An enumeration.

CONSTANT = -1

GROUP_0 = 0

GROUP_1 = 1

GROUP_2 = 2

__module__ = 'ieml.usl.decoration.path'

class ieml.usl.decoration.path.**LexemeIndex**
Bases: *ieml.commons.OrderedEnum*

An enumeration.

CONTENT = 0

FLEXION = 1

__module__ = 'ieml.usl.decoration.path'

class ieml.usl.decoration.path.**LexemePath**(*index: ieml.usl.decoration.path.LexemeIndex*, *child=None*)
Bases: *ieml.usl.decoration.path.UslPath*

USL_TYPE
alias of *ieml.usl.lexeme.Lexeme*

__init__(*index: ieml.usl.decoration.path.LexemeIndex*, *child=None*)
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'ieml.usl.decoration.path'

as_constant(*u=None*)

classmethod build_usl_from_path_to_node(*path_to_node*)

clone(*use_child=False*, *child=None*)

without_morpheme()

class ieml.usl.decoration.path.**PolymorphemePath**(*group_idx: ieml.usl.decoration.path.GroupIndex*, *morpheme: ieml.dictionary.script.script.Script = None*, *multiplicity=None*, *child=None*)
Bases: *ieml.usl.decoration.path.UslPath*

USL_TYPE
alias of *ieml.usl.polymorpheme.PolyMorpheme*

__init__(*group_idx: ieml.usl.decoration.path.GroupIndex*, *morpheme: ieml.dictionary.script.script.Script = None*, *multiplicity=None*, *child=None*)
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'ieml.usl.decoration.path'

as_constant(*u=None*)

classmethod build_usl_from_path_to_node(*path_to_node*)
path_to_node: dict PolymorphemePath -> Script # TODO handle multiplicity :param path_to_node: :return:

clone(*use_child=False*, *child=None*)

without_morpheme()

class ieml.usl.decoration.path.**RolePath**(*role*, *has_focus=False*, *child=None*)
Bases: *ieml.usl.decoration.path.UslPath*

USL_TYPE
alias of *ieml.usl.word.Word*

__init__(*role*, *has_focus=False*, *child=None*)
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'ieml.usl.decoration.path'

as_constant(*u=None*)

classmethod build_usl_from_path_to_node(*path_to_node*)

clone(*use_child=False*, *child=None*)

without_morpheme()

class ieml.usl.decoration.path.**UslPath**(*child=None*)
Bases: object

USL_TYPE
alias of *ieml.usl.usl.USL*

__dict__ = mappingproxy({'__module__': 'ieml.usl.decoration.path', 'USL_TYPE': <class

__eq__(*other*)
Return self==value.

__hash__()
Return hash(self).

__init__(*child=None*)
Initialize self. See help(type(self)) for accurate signature.

__lt__(*other*)
Return self<value.

__module__ = 'ieml.usl.decoration.path'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

as_constant(*u=None*)

classmethod build_usl_from_path_to_node(*path_to_node*)

clone(*use_child=False*, *child=None*)

concat(*suffix: ieml.usl.decoration.path.UslPath*, *force: bool = False*) → ieml.usl.decoration.path.UslPath

contained(*usl*)

deference(*usl: ieml.usl.usl.USL*) → ieml.usl.usl.USL

classmethod from_string(*string*)

has_prefix(*prefix: ieml.usl.decoration.path.UslPath*)

is_constant_path

no_child_clone()

remove_prefix(*prefix: ieml.usl.decoration.path.UslPath*)

split_tail()

tail

without_morpheme()

ieml.usl.decoration.path.**path**(*string*) → ieml.usl.decoration.path.UslPath

ieml.usl.decoration.path.**usl_from_path_values**(*paths_values*)

ieml.usl.decoration.parser.lexer.**get_lexer**(*module=None*)

class ieml.usl.decoration.parser.parser.**PathParser**

Bases: object

__dict__ = mappingproxy({'__module__': 'ieml.usl.decoration.parser.parser', 'tokens':

__init__()

Initialize self. See help(type(self)) for accurate signature.

__module__ = 'ieml.usl.decoration.parser.parser'

__weakref__

list of weak references to the object (if defined)

lock = <unlocked _thread.lock object>

p_error(*p*)

p_flexion_path(*p*)

flexion_path : MORPHEME

p_lexeme_path(*p*)

lexeme_path : LEXEME_POSITION | LEXEME_POSITION SEPARATOR polymorpheme_path | LEXEME_POSITION SEPARATOR flexion_path

p_path(*p*)

path : SEPARATOR | SEPARATOR role_path | SEPARATOR lexeme_path | SEPARATOR flexion_path | SEPARATOR polymorpheme_path

p_polymorpheme_path(*p*)

polymorpheme_path : POLYMORPHEME_POSITION | POLYMORPHEME_POSITION MULTIPLICITY | POLYMORPHEME_POSITION SEPARATOR MORPHEME | POLYMORPHEME_POSITION MULTIPLICITY SEPARATOR MORPHEME

p_role_path(*p*)

role_path : ROLE_TOKEN SEPARATOR role_path_list | ROLE_TOKEN SEPARATOR EXCLAMATION_MARK role_path_list | ROLE_TOKEN SEPARATOR role_path_list SEPARATOR lexeme_path | ROLE_TOKEN SEPARATOR EXCLAMATION_MARK role_path_list SEPARATOR lexeme_path

p_role_path_list(*p*)

role_path_list : role_path_list MORPHEME | MORPHEME | role_path_list ROLE_NAME | ROLE_NAME

parse(*s*)

tokens = ('SEPARATOR', 'ROLE_TOKEN', 'ROLE_NAME', 'ROLE_MORPHEME', 'LEXEME_POSITION',

3.2.5 Literals

class ieml.usl.decoration.instance.**Decoration**(*path: ieml.usl.decoration.path.UslPath*, *value*)

Bases: object

__dict__ = mappingproxy({'__module__': 'ieml.usl.decoration.instance', '__init__': <

__eq__(*other*)
Return self==value.

__hash__ = None

__init__(*path: ieml.usl.decoration.path.UslPath*, *value*)
Initialize self. See help(type(self)) for accurate signature.

__lt__(*other*)
Return self<value.

__module__ = 'ieml.usl.decoration.instance'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

apply(*u: ieml.commons.DecoratedComponent*)

class ieml.usl.decoration.instance.**InstancedUSL**(*u: ieml.usl.usl.USL*, *decorations: List[ieml.usl.decoration.instance.Decoration]*)

Bases: *ieml.usl.usl.USL*

__init__(*u: ieml.usl.usl.USL, decorations: List[ieml.usl.decoration.instance.Decoration]*)
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'ieml.usl.decoration.instance'

__str__()
Return str(self).

check()

do_lt(*other*)

empty

static from_usl(*u: ieml.usl.usl.USL*)

iter_structure()

iter_structure_path(*flexion=False*)

static list_decorations(*u: ieml.usl.usl.USL*, *flexion=False*)

morphemes

syntactic_level = 10

class ieml.usl.decoration.instance.**LiteralContext**

Bases: object

__dict__ = mappingproxy({'__module__': 'ieml.usl.decoration.instance', '__init__': <

__enter__()

__exit__(*exc_type*, *exc_val*, *exc_tb*)

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'ieml.usl.decoration.instance'

__weakref__
list of weak references to the object (if defined)

push(*u*)

ieml.usl.decoration.instance.**literal_context**()

3.3 Utils

ieml.usl.constants.**assert_**(*cond*, *message*)

ieml.usl.constants.**assert_all_in**(*l: List[ieml.dictionary.script.script.Script], _set: Set[ieml.dictionary.script.script.Script], name_l*)

ieml.usl.constants.**assert_atmost_one_from**(*l: List[ieml.dictionary.script.script.Script], _set: Set[ieml.dictionary.script.script.Script], name_l, name_set*)

ieml.usl.constants.**assert_no_one_from**(*l: List[ieml.dictionary.script.script.Script], _set: Set[ieml.dictionary.script.script.Script], name_l, name_set*)

ieml.usl.constants.**assert_only_one_from**(*l: List[ieml.dictionary.script.script.Script], _set: Set[ieml.dictionary.script.script.Script], name_l, name_set*) → ieml.dictionary.script.script.Script

ieml.usl.constants.**check_address_script**(*l: List[ieml.dictionary.script.script.Script], sfun_type*)

ieml.usl.constants.**check_flexion_actant_scripts**(*l: List[ieml.dictionary.script.script.Script], sfun=None*)

ieml.usl.constants.**check_flexion_process_scripts**(*l: List[ieml.dictionary.script.script.Script], sfun=None*)

ieml.usl.constants.**check_flexion_quality**(*l: List[ieml.dictionary.script.script.Script], sfun=None*)

ieml.usl.constants.**check_lexeme_scripts**(*l_pf: List[ieml.dictionary.script.script.Script], l_pc: List[ieml.dictionary.script.script.Script], sfun=None*)

ieml.usl.constants.**class_from_address**(*address*)

class ieml.commons.**DecoratedComponent**(**args*, ***kwargs*)
Bases: object

__dict__ = mappingproxy({'__module__': 'ieml.commons', '__init__': <function Decorat

__init__(**args*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'ieml.commons'

__weakref__
list of weak references to the object (if defined)

clear_literal()

get_literal()

set_literal(*value*)

class ieml.commons.**FolderWatcherCache**(*db_path: str*, *pattern: str*, *cache_folder: str*, *name: str*)

Bases: object

__dict__ = mappingproxy({'__module__': 'ieml.commons', '__init__': <function FolderW

__init__(*db_path: str*, *pattern: str*, *cache_folder: str*, *name: str*)

Cache that check if *folder* content has changed. Compute a hash of the files in the folder and get pruned if the content of this folder change.

Parameters

- **folder** – the folder to watch
- **cache_folder** – the folder to put the cache file

__module__ = 'ieml.commons'

__weakref__

list of weak references to the object (if defined)

cache_file

Returns The cache file absolute path

get() → object

Unpickle and return the object stored in the cache file. :return: the stored object

is_pruned() → bool

Return True if the watched folder content has changed. :return: if the folder content changed

update(*obj*) → None

Update the cache content, remove old cache files from the cache directory.

Parameters **obj** – the object to pickle in the cache

Returns None

class ieml.commons.**LastUpdatedOrderedDict**

Bases: collections.OrderedDict

Store items in the order the keys were last added

__module__ = 'ieml.commons'

__setitem__(*key*, *value*, ***kwargs*)

Set self[key] to value.

class ieml.commons.**OrderedEnum**

Bases: enum.Enum

An enumeration.

__ge__(*other*)

Return self>=value.

__gt__(*other*)

Return self>value.

__le__(*other*)

Return self<=value.

__lt__(*other*)
Return self<value.

__module__ = 'ieml.commons'

class ieml.commons.**Singleton**
Bases: type

__call__(**args*, ***kwargs*)
Call self as a function.

__module__ = 'ieml.commons'

class ieml.commons.**TreeStructure**(**args*, ***kwargs*)
Bases: object

__dict__ = mappingproxy({'__module__': 'ieml.commons', '__init__': <function TreeStr

__eq__(*other*)
Return self==value.

__hash__()
Since the IEML string for any proposition AST is supposed to be unique, it can be used as a hash

__init__(**args*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

__iter__()
Enables the syntactic sugar of iterating directly on an element without accessing "children"

__module__ = 'ieml.commons'

__ne__(*other*)
Return self!=value.

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

tree_iter()

ieml.commons.**cache_results_watch_files**(*path*, *name*)

class ieml.commons.**cached_property**(*factory*)
Bases: object

__dict__ = mappingproxy({'__module__': 'ieml.commons', '__init__': <function cached_

__get__(*instance*, *owner*)

__init__(*factory*)
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'ieml.commons'

__weakref__
list of weak references to the object (if defined)

ieml.commons.**fullname**(*cls*)

ieml.commons.**monitor_decorator**(*name*)

class ieml.constants.**DescriptorsType**
Bases: enum.Enum

An enumeration.

COMMENTS = 'comments'

REFERENTIAL = 'rdf_repositories'

TAGS = 'tags'

TRANSLATIONS = 'translations'

__module__ = 'ieml.constants'

class ieml.constants.**Languages**
Bases: enum.Enum

The language currently supported by the IEML database

EN = 'en'

FR = 'fr'

__module__ = 'ieml.constants'

ieml.constants.**get_iemldb_folder**(*name*)

3.4 Database reference

class ieml.ieml_database.descriptors.**Descriptors**(*df*)
Bases: object

__dict__ = mappingproxy({'__module__': 'ieml.ieml_database.descriptors', '__init__':

__init__(*df*)
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'ieml.ieml_database.descriptors'

__weakref__
list of weak references to the object (if defined)

static from_csv_string(*s*, *assert_unique_ieml=False*)

get_descriptor(*ieml*) → Dict[ieml.constants.DescriptorsType, Dict[ieml.constants.Languages, List[str]]]

get_values(*ieml*, *language*, *descriptor*)

get_values_partial(*ieml*, *language=None*, *descriptor=None*)

ieml.ieml_database.descriptors.**normalize_key**(*ieml*, *key*, *value*, *parse_ieml=False*, *partial=False*, *structure=False*)

class ieml.ieml_database.git_interface.**GitInterface**(*origin='https://github.com/IEMLdev/ieml-language.git'*, *credentials=<pygit2.credentials.Username object>*, *branch='master'*, *commit_id=None*, *folder=None*)
Bases: object

__dict__ = mappingproxy({'__module__': 'ieml.ieml_database.git_interface', '__init__'

__init__(*origin='https://github.com/IEMLdev/ieml-language.git'*, *credentials=<pygit2.credentials.Username object>*, *branch='master'*, *commit_id=None*, *folder=None*)

Parameters

- **origin** –
- **credentials** –
- **branch** – the branch to checkout
- **commit_id** – the commit to checkout
- **folder** –

__module__ = 'ieml.ieml_database.git_interface'

__weakref__
list of weak references to the object (if defined)

add_remote(*name*, *url*)

checkout(*branch=None*, *commit_id=None*)

commit(*signature*, *message*)

current_commit

diff(*commit0*, *commit1*)

get_version()

pull(***kwargs*)

push(***kwargs*)

repo

reset(*commit_id=None*)
Set the current branch HEAD to ref the given commit :param commit_id: if set, reset to this commit id, otherwise to the head of the branch :return: None

status()
ignore path starting with '.'

exception ieml.ieml_database.git_interface.**MergeConflict**(*message*, *conflicts*)
Bases: Exception

__init__(*message*, *conflicts*)
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'ieml.ieml_database.git_interface'

__repr__()
Return repr(self).

__weakref__
list of weak references to the object (if defined)

ieml.ieml_database.git_interface.**get_local_cache_dir**(*origin*)

class ieml.ieml_database.git_interface.**git_transaction**(*db*, *signature*, *message*)
Bases: object

__dict__ = mappingproxy({'__module__': 'ieml.ieml_database.git_interface', '__init__'

__enter__()

__exit__(***kwargs*)

__init__(*db*, *signature*, *message*)
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'ieml.ieml_database.git_interface'

__weakref__
list of weak references to the object (if defined)

class ieml.ieml_database.ieml_database.**IEMLDatabase**(*folder*, *cache_folder=None*, *use_cache=True*)
Bases: object

CLASS_TO_FOLDER = {<class 'ieml.dictionary.script.script.NullScript'>: ('morpheme', 0

HASH_SIZE = 10

MAX_IEML_NAME_SIZE = 100

__dict__ = mappingproxy({'__module__': 'ieml.ieml_database.ieml_database', 'CLASS_TO_

__init__(*folder*, *cache_folder=None*, *use_cache=True*)
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'ieml.ieml_database.ieml_database'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

add_descriptor(*ieml*, *language*, *descriptor*, *value*)

add_structure(*ieml*, *key*, *value*)

escape_value(*v*)

filename_of(*ieml*)

get_descriptors(***kwargs*)

get_dictionary(***kwargs*)

get_list(***kwargs*)

get_structure(***kwargs*)

list(***kwargs*)

path_of(*_ieml*, *descriptor=True*, *mkdir=False*, *normalize=True*)

remove_descriptor(***kwargs*)

remove_structure(*ieml*, *key=None*, *value=None*, *normalize=True*)

class ieml.ieml_database.ieml_database.**Structure**(*df*)
Bases: object

__dict__ = mappingproxy({'__module__': 'ieml.ieml_database.ieml_database', '__init__'

__init__(*df*)
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'ieml.ieml_database.ieml_database'

__weakref__
list of weak references to the object (if defined)

get_values(***kwargs*)

get_values_partial(***kwargs*)

class ieml.ieml_database.transactions.DBTransaction.**DBTransactions**(*gitdb*, *signature*, *cache_folder=None*, *use_cache=True*)

Bases: object

__dict__ = mappingproxy({'__module__': 'ieml.ieml_database.transactions.DBTransaction

__init__(*gitdb*, *signature*, *cache_folder=None*, *use_cache=True*)
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'ieml.ieml_database.transactions.DBTransaction'

__weakref__
list of weak references to the object (if defined)

add_morpheme_paradigm(*script: ieml.dictionary.script.script.Script*, *translations*, *comments*)

create_root_paradigm(*root*, *inhibitions*, *translations*, *comments*)

delete_morpheme_paradigm(*script: ieml.dictionary.script.script.Script*)

delete_morpheme_root_paradigm(*script: ieml.dictionary.script.script.Script*, *empty_descriptors=True*)

set_descriptors(*ieml*, *descriptor*, *value*)

set_inhibitions(*ieml*, *inhibitions*)

update_all_ieml(*f*, *message: str*)

update_morpheme_paradigm(*script_old: ieml.dictionary.script.script.Script*, *script_new: ieml.dictionary.script.script.Script*)

ieml.ieml_database.transactions.DBTransaction.**append_idx_to_dict**(*d*, *idx*)

ieml.**error**(*m*)

Python Module Index

i

Index

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W