CKIPNLP

Release v1.0.2

Mu Yang

May 07, 2021

OVERVIEW

CHAPTER

ONE

INTRODUCTION

1.1 CKIP CoreNLP Toolkit

1.1.1 Features

- Sentence Segmentation
- Word Segmentation
- Part-of-Speech Tagging
- Named-Entity Recognition
- Constituency Parsing
- Coreference Resolution

1.1.2 Git

https://github.com/ckiplab/ckipnlp

1.1.3 PyPI

https://pypi.org/project/ckipnlp

1.1.4 Documentation

https://ckipnlp.readthedocs.io/

1.1.5 Online Demo

https://ckip.iis.sinica.edu.tw/service/corenlp

1.1.6 Contributers

- Mu Yang at CKIP (Author & Maintainer)
- Wei-Yun Ma at CKIP (Maintainer)
- DouglasWu

1.2 Installation

1.2.1 Requirements

- Python 3.6+
- TreeLib 1.5+
- CkipTagger 0.2.1+ [Optional, Recommended]
- CkipClassic 1.0+ [Optional, Recommended]
- TensorFlow / TensorFlow-GPU 1.13.1+ [Required by CkipTagger]

1.2.2 Driver Requirements

Driver	Built-in	CkipTagger	CkipClassic
Sentence Segmentation	✓		
Word Segmentation†		✓	✓
Part-of-Speech Tagging†		✓	✓
Constituency Parsing			✓
Named-Entity Recognition		✓	
Coreference Resolution‡	✓	✓	✓

- † These drivers require only one of either backends.
- ‡ Coreference implementation does not require any backend, but requires results from word segmentation, part-of-speech tagging, constituency parsing, and named-entity recognition.

1.2.3 Installation via Pip

- No backend (not recommended): `pip install ckipnlp`.
- With CkipTagger backend (recommended): `pip install ckipnlp[tagger]` or `pip install ckipnlp[tagger-gpu]`.
- With CkipClassic Parser Client backend (recommended): `pip install ckipnlp[classic]`.
- With CkipClassic offline backend: Please refer https://ckip-classic.readthedocs.io/en/latest/main/readme.html#installation for CkipClassic installation guide.

Attention: To use CkipClassic Parser Client backend, please

1. Register an account at http://parser.iis.sinica.edu.tw/v1/reg.exe
2. Set the username and password in the pipeline's options:

```
pipeline = CkipPipeline(opts={'con_parser': {'username': YOUR_USERNAME, 'password':␣
↪YOUR_PASSWORD})
```

1.3 Detail

See https://ckipnlp.readthedocs.io/ for full documentation.

1.4 License

CHAPTER

TWO

USAGE

CkipNLP provides a set of human language technology tools, including

- Sentence Segmentation
- Word Segmentation
- Part-of-Speech Tagging
- Named-Entity Recognition
- Constituency Parsing
- Coreference Resolution

The library is build around three types of classes:

- Containers such as *SegParagraph* are the basic data structures for inputs and outputs.
- Drivers such as *CkipTaggerWordSegmenter* that apply specific tool on the inputs.
- Pipelines such as *CkipPipeline* are collections of drivers that automatically handles the dependencies between inputs and outputs.

2.1 Containers

2.1.1 Containers Prototypes

All the container objects can be convert from/to other formats:

- `from_text()`, `to_text()` for plain-text conversions;
- `from_list()`, `to_list()` for list-like python object conversions;
- `from_dict()`, `to_dict()` for dictionary-like python object (key-value mappings) conversions;
- `from_json()`, `to_json()` for JSON format conversions (based-on dictionary-like format conversions).

Here are the interfaces, where `CONTAINER_CLASS` refers to the container class.

```
obj = CONTAINER_CLASS.from_text(plain_text)
plain_text = obj.to_text()

obj = CONTAINER_CLASS.from_list([ value1, value2 ])
list_obj = obj.to_list()

obj = CONTAINER_CLASS.from_dict({ key: value })
dict_obj = obj.to_dict()
```

(continues on next page)

(continued from previous page)

```
obj = CONTAINER_CLASS.from_json(json_str)
json_str = obj.to_json()
```

Note that not all container provide all above conversions. Here is the table of implemented methods. Please refer the documentation of each container for format details.

Container	Item	from/to text	from/to list, dict, json
TextParagraph	str	✓	✓
SegSentence	str	✓	✓
SegParagraph	*SegSentence*	✓	✓
NerToken			✓
NerSentence	*NerToken*		✓
NerParagraph	*NerSentence*		✓
ParseClause		only to	✓
ParseSentence	*ParseClause*	only to	✓
ParseParagraph	*ParseSentence*	only to	✓
CorefToken		only to	✓
CorefSentence	*CorefToken*	only to	✓
CorefParagraph	*CorefSentence*	only to	✓

2.1.2 WS with POS

There are also conversion routines for word-segmentation and part-of-speech containers jointly. For example, *WsPosToken* provides routines for a word (str) with POS-tag (str):

```
ws_obj, pos_obj = WsPosToken.from_text('(Na)')
plain_text = WsPosToken.to_text(ws_obj, pos_obj)

ws_obj, pos_obj = WsPosToken.from_list([ '', 'Na' ])
list_obj = WsPosToken.to_list(ws_obj, pos_obj)

ws_obj, pos_obj = WsPosToken.from_dict({ 'word': '', 'pos': 'Na', })
dict_obj = WsPosToken.to_dict(ws_obj, pos_obj)

ws_obj, pos_obj = WsPosToken.from_json(json_str)
json_str = WsPosToken.to_json(ws_obj, pos_obj)
```

Similarly, *WsPosSentence*/*WsPosParagraph* provides routines for word-segmented and POS sentence/paragraph (*SegSentence*/*SegParagraph*) respectively.

2.1.3 Parse Tree

In addition to *ParseClause*, there are also tree utilities base on TreeLib.

ParseTree is the tree structure of a parse clause. One may use `from_text()` and `to_text()` for plain-text conversion; `from_dict()`,`to_dict()` for dictionary-like object conversion; and also `from_json()`,`to_json()` for JSON string conversion.

ParseTree also provide `from_penn()` and `to_penn()` methods for Penn Treebank conversion. One may use `to_penn()` together with SvgLing to generate SVG tree graphs.

ParseTree is a TreeLib tree with *ParseNode* as its nodes. The data of these nodes is stored in a *ParseNodeData* (accessed by `node.data`), which is a tuple of `role` (semantic role), `pos` (part-of-speech tagging), `word`.

ParseTree provides useful methods: `get_heads()` finds the head words of the clause; `get_relations()` extracts all relations in the clause; `get_subjects()` returns the subjects of the clause.

```
from ckipnlp.container import ParseClause, ParseTree

#
clause = ParseClause(
↪'S(goal:NP(possessor:N(head:Nhaa:|Head:DE:)|Head:Nab(DUMMY1:Nab(DUMMY1:Nab:|Head:Caa:|DUMMY2:Naa:)
↪')

tree = clause.to_tree()

print('Show Tree')
tree.show()

print('Get Heads of {}'.format(tree[5]))
print('-- Semantic --')
for head in tree.get_heads(5, semantic=True): print(repr(head))
print('-- Syntactic --')
for head in tree.get_heads(5, semantic=False): print(repr(head))
print()

print('Get Relations of {}'.format(tree[0]))
print('-- Semantic --')
for rel in tree.get_relations(0, semantic=True): print(repr(rel))
print('-- Syntactic --')
for rel in tree.get_relations(0, semantic=False): print(repr(rel))
print()

#
tree_text =
↪'S(theme:NP(DUMMY1:NP(Head:Nhaa:)|Head:Caa:|DUMMY2:NP(Head:Naa:))|evaluation:Dbb:|quantity:Dab:|deg
↪'

tree = ParseTree.from_text(tree_text)

print('Show Tree')
tree.show()

print('Get get_subjects of {}'.format(tree[0]))
print('-- Semantic --')
for subject in tree.get_subjects(0, semantic=True): print(repr(subject))
print('-- Syntactic --')
for subject in tree.get_subjects(0, semantic=False): print(repr(subject))
print()
```

2.2 Drivers

class Driver(*, *lazy=False*, ...)
The prototype of CkipNLP Drivers.

Parameters **lazy** (*bool*) – Lazy initialize the driver. (Call *init()* at the first call of *__call__()* instead.)

driver_type: str
The type of this driver.

driver_family: str
The family of this driver.

driver_inputs: Tuple[str, ...]
The inputs of this driver.

init()
Initialize the driver (by calling the `_init()` function).

__call__(*, ...)
Call the driver (by calling the `_call()` function).

Here are the list of the drivers:

Driver Type \ Family	'default'	'tagger'	'classic'	'classic-client'
Sentence Segmenter	*CkipSentenceSegmenter*			
Word Segmenter		*CkipTaggerWordSegmenter*	*CkipClassicWordSegmenter*†	
Pos Tagger		*CkipTaggerPosTagger*	*CkipClassicWordSegmenter*†	
Ner Chunker		*CkipTaggerNerChunker*		
Constituency Parser			*CkipClassicConParser*	*CkipClassicConParserClient*‡
Coref Chunker	*CkipCorefChunker*			

- † Not compatible with *CkipCorefPipeline*.
- ‡ Please register an account at http://parser.iis.sinica.edu.tw/v1/reg.exe and set the environment variables `$CKIPPARSER_USERNAME` and `$CKIPPARSER_PASSWORD`.

2.3 Pipelines

2.3.1 Kernel Pipeline

The *CkipPipeline* connect drivers of sentence segmentation, word segmentation, part-of-speech tagging, named-entity recognition, and sentence parsing.

The *CkipDocument* is the workspace of *CkipPipeline* with input/output data. Note that *CkipPipeline* will store the result into *CkipDocument* in-place.

The *CkipPipeline* will compute all necessary dependencies. For example, if one calls `get_ner()` with only raw-text input, the pipeline will automatically calls `get_text()`, `get_ws()`, `get_pos()`.

```
from ckipnlp.pipeline import CkipPipeline, CkipDocument

pipeline = CkipPipeline()
doc = CkipDocument(raw='')

# Word Segmentation
pipeline.get_ws(doc)
print(doc.ws)
for line in doc.ws:
    print(line.to_text())

# Part-of-Speech Tagging
pipeline.get_pos(doc)
print(doc.pos)
for line in doc.pos:
    print(line.to_text())

# Named-Entity Recognition
pipeline.get_ner(doc)
print(doc.ner)

# Constituency Parsing
pipeline.get_conparse(doc)
print(doc.conparse)

################################################################

from ckipnlp.container.util.wspos import WsPosParagraph

# Word Segmentation & Part-of-Speech Tagging
for line in WsPosParagraph.to_text(doc.ws, doc.pos):
    print(line)
```

To customize the driver (e.g. disable CUDA in *CkipTaggerWordSegmenter*), you may pass the options to the pipeline:

```
pipeline = CkipPipeline(opts = {'word_segmenter': {'disable_cuda': True}})
```

Please refer each driver's documentation for the extra options.

2.3.2 Co-Reference Pipeline

The *CkipCorefPipeline* is a extension of *CkipPipeline* by providing coreference resolution. The pipeline first do named-entity recognition as *CkipPipeline* do, followed by alignment algorithms to fix the word-segmentation and part-of-speech tagging outputs, and then do coreference resolution based sentence parsing result.

The *CkipCorefDocument* is the workspace of *CkipCorefPipeline* with input/output data. Note that *CkipCorefDocument* will store the result into *CkipCorefPipeline*.

```
from ckipnlp.pipeline import CkipCorefPipeline, CkipDocument

pipeline = CkipCorefPipeline()
doc = CkipDocument(raw='')
```

(continues on next page)

(continued from previous page)

```
# Co-Reference
corefdoc = pipeline(doc)
print(corefdoc.coref)
for line in corefdoc.coref:
    print(line.to_text())
```

CHAPTER

THREE

TABLES OF TAGS

3.1 Part-of-Speech Tags

Tag	Description
A	
Caa	
Cab	
Cba	
Cbb	
D	
Da	
Dfa	
Dfb	
Di	
Dk	
DM	
I	
Na	
Nb	
Nc	
Ncd	
Nd	
Nep	
Neqa	
Neqb	
Nes	
Neu	
Nf	
Ng	
Nh	
Nv	
P	
T	
VA	
VAC	
VB	
VC	
VCL	

continues on next page

Table 1 – continued from previous page

Tag	Description
VD	
VF	
VE	
VG	
VH	
VHC	
VI	
VJ	
VK	
VL	
V_2	
DE	
SHI	
FW	
COLONCATEGORY	
COMMACATEGORY	
DASHCATEGORY	
DOTCATEGORY	
ETCCATEGORY	
EXCLAMATIONCATEGORY	
PARENTHESISCATEGORY	
PAUSECATEGORY	
PERIODCATEGORY	
QUESTIONCATEGORY	
SEMICOLONCATEGORY	
SPCHANGECATEGORY	
WHITESPACE	

3.2 Constituency Parsing Tags

Tag	Description
S	SNP
VP	V
NP	N
GP	NgDUMMY1
PP	PDUMMY
XP	CXXPVPVPNP
DM	

3.3 Constituency Parsing Roles

Role	Description
#	
apposition	
possessor	
predication	
property	
quantifier	
#–	
agent	
benefactor	
causer	
companion	
comparison	
experiencer	
goal	
range	
source	
target	
theme	
topic	
#–	
aspect	
degree	
deixis	
deontics	
duration	
evaluation	
epistemics	
frequency	
instrument	
interjection	
location	
manner	
negation	
particle	
quantity	
standard	
time	
#–	
addition	
alternative	
avoidance	
complement	
conclusion	
condition	
concession	
contrast	
conversion	

continues on next page

Table 2 – continued from previous page

Role	Description
exclusion	
hypothesis	
listing	
purpose	
reason	
rejection	
result	
restriction	
selection	
uncondition	
whatever	
#	
DUMMY	
DUMMY1	
DUMMY2	
Head	Head
head	
nominal	

CHAPTER

FOUR

CKIPNLP PACKAGE

The Official CKIP CoreNLP Toolkit.

Subpackages

4.1 ckipnlp.container package

This module implements specialized container datatypes for CKIPNLP.

Subpackages

4.1.1 ckipnlp.container.util package

This module implements specialized utilities for CKIPNLP containers.

Submodules

ckipnlp.container.util.parse_tree module

This module provides tree containers for parsed sentences.

class ckipnlp.container.util.parse_tree.**ParseNodeData**(*role: Optional[str] = None*, *pos: Optional[str] = None*, *word: Optional[str] = None*)

Bases: *ckipnlp.container.base.BaseTuple*, ckipnlp.container.util.parse_tree._ParseNodeData

A parse node.

Variables

- **role** (*str*) – the semantic role.
- **pos** (*str*) – the POS-tag.
- **word** (*str*) – the text term.

Note: This class is an subclass of `tuple`. To change the attribute, please create a new instance instead.

Data Structure Examples

Text format Used for *from_text()* and `to_text()`.

```
'Head:Na:'  # role / POS-tag / text-term
```

List format Not implemented.

Dict format Used for `from_dict()` and `to_dict()`.

```
{
    'role': 'Head',    # role
    'pos': 'Na',       # POS-tag
    'word': '',  # text term
}
```

---

classmethod from_text(*data*)

Construct an instance from text format.

Parameters **data** (*str*) – text such as `'Head:Na:'`.

Note:

- `'Head:Na:'` -> **role** = `'Head'`, **pos** = `'Na'`, **word** = `''`
- `'Head:Na'` -> **role** = `'Head'`, **pos** = `'Na'`, **word** = `None`
- `'Na'` -> **role** = `None`, **pos** = `'Na'`, **word** = `None`

---

class ckipnlp.container.util.parse_tree.**ParseNode**(*tag=None*, *identifier=None*, *expanded=True*, *data=None*)

Bases: *ckipnlp.container.base.Base*, `treelib.node.Node`

A parse node for tree.

Variables **data** (*ParseNodeData*) –

See also:

treelib.tree.Node Please refer https://treelib.readthedocs.io/ for built-in usages.

Data Structure Examples

Text format Not implemented.

List format Not implemented.

Dict format Used for `to_dict()`.

```
{
    'role': 'Head',    # role
    'pos': 'Na',       # POS-tag
    'word': '',  # text term
}
```

---

data_class

alias of *ckipnlp.container.util.parse_tree.ParseNodeData*

class ckipnlp.container.util.parse_tree.**ParseRelation**(*head:* ckipnlp.container.util.parse_tree.ParseNode, *tail:* ckipnlp.container.util.parse_tree.ParseNode, *relation:* ckipnlp.container.util.parse_tree.ParseNode)

Bases: *ckipnlp.container.base.Base*, ckipnlp.container.util.parse_tree._ParseRelation

A parse relation.

Variables

- **head** (*ParseNode*) – the head node.
- **tail** (*ParseNode*) – the tail node.
- **relation** (*ParseNode*) – the relation node. (the semantic role of this node is the relation.)

Notes

The parent of the relation node is always the common ancestor of the head node and tail node.

Data Structure Examples

Text format Not implemented.

List format Not implemented.

Dict format Used for `to_dict()`.

```
{
    'tail': { 'role': 'Head', 'pos': 'Nab', 'word': '' }, # head node
    'tail': { 'role': 'particle', 'pos': 'Td', 'word': '' }, # tail node
    'relation': 'particle',  # relation
}
```

---

class ckipnlp.container.util.parse_tree.**ParseTree**(*tree=None*, *deep=False*, *node_class=None*, *identifier=None*)

Bases: *ckipnlp.container.base.Base*, treelib.tree.Tree

A parse tree.

See also:

treereelib.tree.Tree Please refer https://treelib.readthedocs.io/ for built-in usages.

Data Structure Examples

Text format Used for *from_text()* and *to_text()*.

```
'S(Head:Nab:|particle:Td:)'
```

List format Not implemented.

Dict format Used for *from_dict()* and *to_dict()*. A dictionary such as `{ 'id': 0, 'data': { ... }, 'children': [ ... ] }`, where `'data'` is a dictionary with the same format as `ParseNodeData.to_dict()`, and `'children'` is a list of dictionaries of subtrees with the same format as this tree.

```
{
    'id': 0,
    'data': {
        'role': None,
        'pos': 'S',
        'word': None,
    },
    'children': [
        {
            'id': 1,
            'data': {
                'role': 'Head',
                'pos': 'Nab',
                'word': '',
            },
            'children': [],
        },
        {
            'id': 2,
            'data': {
                'role': 'particle',
                'pos': 'Td',
                'word': '',
            },
            'children': [],
        },
    ],
}
```

Penn Treebank format Used for *from_penn()* and *to_penn()*.

```
[
    'S',
    [ 'Head:Nab', '', ],
    [ 'particle:Td', '', ],
]
```

Note: One may use *to_penn()* together with SvgLing to generate SVG tree graphs.

---

node_class
alias of *ckipnlp.container.util.parse_tree.ParseNode*

classmethod from_text(*data*)
Construct an instance from text format.

Parameters **data** (*str*) – A parse tree in text format (*ParseClause.clause*).

See also:

ParseClause.to_tree().

to_text(*node_id=None*)
Transform to plain text.

Parameters **node_id** (*int*) – Output the plain text format for the subtree under **node_id**.

Returns *str*

classmethod from_dict(*data*)
Construct an instance from python built-in containers.

Parameters **data** (*str*) – A parse tree in dictionary format.

to_dict(*node_id=None*)
Transform to python built-in containers.

Parameters **node_id** (*int*) – Output the plain text format for the subtree under **node_id**.

Returns *str*

classmethod from_penn(*data*)
Construct an instance from Penn Treebank format.

to_penn(*node_id=None*, *, *with_role=True*, *with_word=True*, *sep=':'*)
Transform to Penn Treebank format.

Parameters

- **node_id** (*int*) – Output the plain text format for the subtree under **node_id**.
- **with_role** (*bool*) – Contains role-tag or not.
- **with_word** (*bool*) – Contains word or not.
- **sep** (*str*) – The seperator between role and POS-tag.

Returns *list*

show(*, *key=<function ParseTree.<lambda>>*, *idhidden=False*, **kwargs*)
Show pretty tree.

get_children(*node_id*, *, *role*)
Get children of a node with given role.

Parameters

- **node_id** (*int*) – ID of target node.
- **role** (*str*) – the target role.

Yields *ParseNode* – the children nodes with given role.

get_heads(*root_id=None*, *, *semantic=True*, *deep=True*)
Get all head nodes of a subtree.

Parameters

- **root_id** (*int*) – ID of the root node of target subtree.
- **semantic** (*bool*) – use semantic/syntactic policy. For semantic mode, return `DUMMY` or `head` instead of syntactic `Head`.
- **deep** (*bool*) – find heads recursively.

Yields *ParseNode* – the head nodes.

get_relations(*root_id=None*, *, *semantic=True*)
Get all relations of a subtree.

Parameters

- **root_id** (*int*) – ID of the subtree root node.
- **semantic** (*bool*) – please refer *get_heads()* for policy detail.

Yields *ParseRelation* – the relations.

get_subjects(*root_id=None*, *, *semantic=True*, *deep=True*)
Get the subject node of a subtree.

Parameters

- **root_id** (*int*) – ID of the root node of target subtree.
- **semantic** (*bool*) – please refer *get_heads()* for policy detail.
- **deep** (*bool*) – please refer *get_heads()* for policy detail.

Yields *ParseNode* – the subject node.

Notes

A node can be a subject if either:

1. is a head of *NP*
2. is a head of a subnode (*N*) of *S* with subject role
3. is a head of a subnode (*N*) of *S* with neutral role and before the head (*V*) of *S*

---

ckipnlp.container.util.wspos module

This module provides containers for word-segmented sentences with part-of-speech-tags.

class ckipnlp.container.util.wspos.**WsPosToken**(*word: Optional[str] = None*, *pos: Optional[str] = None*)
Bases: *ckipnlp.container.base.BaseTuple*, ckipnlp.container.util.wspos._WsPosToken

A word with POS-tag.

Variables

- **word** (*str*) – the word.
- **pos** (*str*) – the POS-tag.

Note: This class is an subclass of *tuple*. To change the attribute, please create a new instance instead.

Data Structure Examples

Text format Used for *from_text()* and `to_text()`.

```
'(Na)'  # word / POS-tag
```

List format Used for `from_list()` and `to_list()`.

```
[
    '', # word
    'Na',     # POS-tag
]
```

Dict format Used for `from_dict()` and `to_dict()`.

```
{
    'word': '', # word
    'pos': 'Na',      # POS-tag
}
```

---

classmethod from_text(*data*)
Construct an instance from text format.

Parameters **data** (*str*) – text such as `'(Na)'`.

Note:

- `'(Na)'` -> word = `''`, pos = `'Na'`
- `''` -> word = `''`, pos = `None`

---

class ckipnlp.container.util.wspos.**WsPosSentence**
Bases: `object`

A helper class for data conversion of word-segmented and part-of-speech sentences.

classmethod from_text(*data*)
Convert text format to word-segmented and part-of-speech sentences.

Parameters **data** (*str*) – text such as `'(Na)\u3000(T)'`.

Returns

- *SegSentence* – the word sentence
- *SegSentence* – the POS-tag sentence.

static to_text(*word*, *pos*)
Convert text format to word-segmented and part-of-speech sentences.

Parameters

- **word** (*SegSentence*) – the word sentence
- **pos** (*SegSentence*) – the POS-tag sentence.

Returns *str* – text such as `'(Na)\u3000(T)'`.

class ckipnlp.container.util.wspos.**WsPosParagraph**
Bases: `object`

A helper class for data conversion of word-segmented and part-of-speech sentence lists.

classmethod from_text(*data*)
Convert text format to word-segmented and part-of-speech sentence lists.

Parameters **data** (*Sequence[str]*) – list of sentences such as `'(Na)\u3000(T)'`.

Returns

- *SegParagraph* – the word sentence list
- *SegParagraph* – the POS-tag sentence list.

static to_text(*word*, *pos*)
Convert text format to word-segmented and part-of-speech sentence lists.

Parameters

- **word** (*SegParagraph*) – the word sentence list
- **pos** (*SegParagraph*) – the POS-tag sentence list.

Returns *List[str]* – list of sentences such as `'(Na)\u3000(T)'`.

Submodules

4.1.2 ckipnlp.container.base module

This module provides base containers.

class ckipnlp.container.base.**Base**
Bases: object

The base CKIPNLP container.

abstract classmethod from_text(*data*)
Construct an instance from text format.

Parameters **data** (*str*) –

abstract to_text()
Transform to plain text.

Returns *str*

abstract classmethod from_list(*data*)
Construct an instance from python built-in containers.

abstract to_list()
Transform to python built-in containers.

abstract classmethod from_dict(*data*)
Construct an instance from python built-in containers.

abstract to_dict()
Transform to python built-in containers.

classmethod from_json(*data*, ***kwargs*)
Construct an instance from JSON format.

Parameters **data** (*str*) – please refer *from_dict()* for format details.

to_json(***kwargs*)
Transform to JSON format.

Returns *str*

class ckipnlp.container.base.**BaseTuple**
Bases: *ckipnlp.container.base.Base*

The base CKIPNLP tuple.

classmethod from_list(*data*)
Construct an instance from python built-in containers.

Parameters **data** (*list*) –

to_list()
Transform to python built-in containers.

Returns *list*

classmethod from_dict(*data*)
Construct an instance from python built-in containers.

Parameters **data** (*dict*) –

to_dict()
Transform to python built-in containers.

Returns *dict*

class ckipnlp.container.base.**BaseList**(*initlist=None*)
Bases: ckipnlp.container.base._BaseList, ckipnlp.container.base._InterfaceItem

The base CKIPNLP list.

item_class = Not Implemented
Must be a CKIPNLP container class.

class ckipnlp.container.base.**BaseList0**(*initlist=None*)
Bases: ckipnlp.container.base._BaseList, ckipnlp.container.base._InterfaceBuiltInItem

The base CKIPNLP list with built-in item class.

item_class = Not Implemented
Must be a built-in type.

class ckipnlp.container.base.**BaseSentence**(*initlist=None*)
Bases: ckipnlp.container.base._BaseSentence, ckipnlp.container.base._InterfaceItem

The base CKIPNLP sentence.

item_class = Not Implemented
Must be a CKIPNLP container class.

class ckipnlp.container.base.**BaseSentence0**(*initlist=None*)
Bases: ckipnlp.container.base._BaseSentence, ckipnlp.container.base._InterfaceBuiltInItem

The base CKIPNLP sentence with built-in item class.

item_class = Not Implemented
Must be a built-in type.

4.1.3 ckipnlp.container.coref module

This module provides containers for coreference sentences.

class ckipnlp.container.coref.**CorefToken**(*word*, *coref*, *idx*, ***kwargs*)

Bases: *ckipnlp.container.base.BaseTuple*, ckipnlp.container.coref._CorefToken

A coreference token.

Variables

- **word** (*str*) – the token word.
- **coref** (*Tuple[int, str]*) – the coreference ID and type. *None* if not a coreference source or target.
  - **type:**
    - *'source'*: coreference source.
    - *'target'*: coreference target.
    - *'zero'*: null element coreference target.
- **idx** (*Tuple[int, int]*) – the node indexes (clause index, token index) in parse tree. **idx[1]** = *None* if this node is a null element or the punctuations.

Note: This class is an subclass of `tuple`. To change the attribute, please create a new instance instead.

Data Structure Examples

Text format Used for `to_text()`.

```
'_0'
```

List format Used for `from_list()` and `to_list()`.

```
[
    '',       # token word
    (0, 'source'), # coref ID and type
    (2, 2),        # node index
]
```

Dict format Used for `from_dict()` and `to_dict()`.

```
{
    'word': '',        # token word
    'coref': (0, 'source'), # coref ID and type
    'idx': (2, 2),          # node index
}
```

---

class ckipnlp.container.coref.**CorefSentence**(*initlist=None*)

Bases: *ckipnlp.container.base.BaseSentence*

A list of coreference sentence.

Data Structure Examples

Text format Used for `to_text()`.

```
# Token segmented by \u3000 (full-width space)
'_0_0'
```

List format Used for `from_list()` and `to_list()`.

```
[
    [ '', None, (0, None,), ],
    [ '', None, (1, 0,), ],
    [ '', None, (1, 1,), ],
    [ '', None, (1, None,), ],
    [ '', (0, 'source'), (2, 2,), ],
    [ '', (0, 'target'), (2, 3,), ],
    [ '', None, (2, 4,), ],
]
```

Dict format Used for `from_dict()` and `to_dict()`.

```
[
    { 'word': '', 'coref': None, 'idx': (0, None,), },
    { 'word': '', 'coref': None, 'idx': (1, 0,), },
    { 'word': '', 'coref': None, 'idx': (1, 1,), },
    { 'word': '', 'coref': None, 'idx': (1, None,), },
    { 'word': '', 'coref': (0, 'source'), 'idx': (2, 2,), },
    { 'word': '', 'coref': (0, 'target'), 'idx': (2, 3,), },
    { 'word': '', 'coref': None, 'idx': (2, 4,), },
]
```

---

item_class
alias of *ckipnlp.container.coref.CorefToken*

class ckipnlp.container.coref.**CorefParagraph**(*initlist=None*)
Bases: *ckipnlp.container.base.BaseList*

A list of coreference sentence.

Data Structure Examples

Text format Used for `to_text()`.

```
[
    '_0_0', # Sentence 1
    'None_0', # Sentence 1
]
```

List format Used for `from_list()` and `to_list()`.

```
[
    [ # Sentence 1
        [ '', None, (0, None,), ],
        [ '', None, (1, 0,), ],
        [ '', None, (1, 1,), ],
        [ '', None, (1, None,), ],
        [ '', (0, 'source'), (2, 2,), ],
        [ '', (0, 'target'), (2, 3,), ],
        [ '', None, (2, 4,), ],
```

(continues on next page)

(continued from previous page)

```
    ],
    [ # Sentence 2
        [ '', None, (0, 1,), ],
        [ None, (0, 'zero'), (0, None,), ],
        [ '', None, (0, 2,), ],
        [ '', None, (0, 3,), ],
        [ '', None, (0, 5,), ],
    ],
]
```

Dict format Used for `from_dict()` and `to_dict()`.

```
[
    [ # Sentence 1
        { 'word': '', 'coref': None, 'idx': (0, None,), },
        { 'word': '', 'coref': None, 'idx': (1, 0,), },
        { 'word': '', 'coref': None, 'idx': (1, 1,), },
        { 'word': '', 'coref': None, 'idx': (1, None,), },
        { 'word': '', 'coref': (0, 'source'), 'idx': (2, 2,), },
        { 'word': '', 'coref': (0, 'target'), 'idx': (2, 3,), },
        { 'word': '', 'coref': None, 'idx': (2, 4,), },
    ],
    [ # Sentence 2
        { 'word': '', 'coref': None, 'idx': (0, 1,), },
        { 'word': None, 'coref': (0, 'zero'), 'idx': (0, None,), },
        { 'word': '', 'coref': None, 'idx': (0, 2,), },
        { 'word': '', 'coref': None, 'idx': (0, 3,), },
        { 'word': '', 'coref': None, 'idx': (0, 5,), },
    ],
]
```

---

item_class
alias of *ckipnlp.container.coref.CorefSentence*

4.1.4 ckipnlp.container.ner module

This module provides containers for NER sentences.

class ckipnlp.container.ner.**NerToken**(*word*, *ner*, *idx*, ***kwargs*)
Bases: *ckipnlp.container.base.BaseTuple*, ckipnlp.container.ner._NerToken

A named-entity recognition token.

Variables

- **word** (*str*) – the token word.
- **ner** (*str*) – the NER-tag.
- **idx** (*Tuple[int, int]*) – the starting / ending index.

Note: This class is an subclass of `tuple`. To change the attribute, please create a new instance instead.

Data Structure Examples

Text format Not implemented

List format Used for `from_list()` and `to_list()`.

```
[
    ''      # token word
    'LANGUAGE', # NER-tag
    (0, 3),      # starting / ending index.
]
```

Dict format Used for `from_dict()` and `to_dict()`.

```
{
    'word': '',    # token word
    'ner': 'LANGUAGE', # NER-tag
    'idx': (0, 3),      # starting / ending index.
}
```

CkipTagger format Used for *from_tagger()* and *to_tagger()*.

```
(
    0,          # starting index
    3,          # ending index
    'LANGUAGE', # NER-tag
    '',     # token word
)
```

---

classmethod from_tagger(*data*)
Construct an instance from CkipTagger format.

to_tagger()
Transform to CkipTagger format.

class ckipnlp.container.ner.**NerSentence**(*initlist=None*)
Bases: *ckipnlp.container.base.BaseSentence*

A named-entity recognition sentence.

Data Structure Examples

Text format Not implemented

List format Used for `from_list()` and `to_list()`.

```
[
    [ '', 'GPE', (0, 2), ],    # name-entity 1
    [ '', 'ORG', (3, 5), ], # name-entity 2
]
```

Dict format Used for `from_dict()` and `to_dict()`.

```
[
    { 'word': '', 'ner': 'GPE', 'idx': (0, 2), },    # name-entity 1
    { 'word': '', 'ner': 'ORG', 'idx': (3, 5), }, # name-entity 2
]
```

CkipTagger format Used for *from_tagger()* and *to_tagger()*.

```
[
    ( 0, 2, 'GPE', '', ),    # name-entity 1
    ( 3, 5, 'ORG', '', ), # name-entity 2
]
```

---

item_class
alias of *ckipnlp.container.ner.NerToken*

classmethod from_tagger(*data*)
Construct an instance from CkipTagger format.

to_tagger()
Transform to CkipTagger format.

class ckipnlp.container.ner.**NerParagraph**(*initlist=None*)
Bases: *ckipnlp.container.base.BaseList*

A list of named-entity recognition sentence.

Data Structure Examples

Text format Not implemented

List format Used for `from_list()` and `to_list()`.

```
[
    [ # Sentence 1
        [ '', 'LANGUAGE', (0, 3), ],
    ],
    [ # Sentence 2
        [ '', 'GPE', (0, 2), ],
        [ '', 'ORG', (3, 5), ],
    ],
]
```

Dict format Used for `from_dict()` and `to_dict()`.

```
[
    [ # Sentence 1
        { 'word': '', 'ner': 'LANGUAGE', 'idx': (0, 3), },
    ],
    [ # Sentence 2
        { 'word': '', 'ner': 'GPE', 'idx': (0, 2), },
        { 'word': '', 'ner': 'ORG', 'idx': (3, 5), },
    ],
]
```

CkipTagger format Used for *from_tagger()* and *to_tagger()*.

```
[
    [ # Sentence 1
        ( 0, 3, 'LANGUAGE', '', ),
    ],
    [ # Sentence 2
        ( 0, 2, 'GPE', '', ),
        ( 3, 5, 'ORG', '', ),
    ],
]
```

item_class
alias of *ckipnlp.container.ner.NerSentence*

classmethod from_tagger(*data*)
Construct an instance from CkipTagger format.

to_tagger()
Transform to CkipTagger format.

4.1.5 ckipnlp.container.parse module

This module provides containers for parsed sentences.

class ckipnlp.container.parse.**ParseClause**(*clause: Optional[str] = None*, *delim: str = ''*)
Bases: *ckipnlp.container.base.BaseTuple*, ckipnlp.container.parse._ParseClause

A parse clause.

Variables

- **clause** (*str*) – the parse clause.
- **delim** (*str*) – the punctuations after this clause.

Note: This class is an subclass of `tuple`. To change the attribute, please create a new instance instead.

Data Structure Examples

Text format Used for `to_text()`.

```
'S(Head:Nab:|particle:Td:)' # delim are ignored
```

List format Used for `from_list()`, and `to_list()`.

```
[
    'S(Head:Nab:|particle:Td:)', # parse clause
    '',                          # punctuations
]
```

Dict format Used for `from_dict()` and `to_dict()`.

```
{
    'clause': 'S(Head:Nab:|particle:Td:)', # parse clause
    'delim': '',                           # punctuations
}
```

to_tree()
Transform to tree format.

Returns *ParseTree* – the tree format of this clause. (*None* if **clause** is *None*)

See also:

ParseTree.from_text().

class ckipnlp.container.parse.**ParseSentence**(*initlist=None*)
Bases: *ckipnlp.container.base.BaseList*

A parse sentence.

Data Structure Examples

Text format Used for `to_text()`.

```
[ # delim are ignored
    'S(Head:Nab:|particle:Td:)',                    # Clause 1
    '%(particle:I:|manner:Dh:|manner:Dh:|time:Dh:), # Clause 2
]
```

List format Used for `from_list()`, and `to_list()`.

```
[
    [ # Clause 1
        'S(Head:Nab:|particle:Td:)',
        '',
    ],
    [ # Clause 2
        '%(particle:I:|manner:Dh:|manner:Dh:|time:Dh:),
        '',
    ],
]
```

Dict format Used for `from_dict()` and `to_dict()`.

```
[
    { # Clause 1
        'clause': 'S(Head:Nab:|particle:Td:)',
        'delim': '',
    },
    { # Clause 2
        'clause': '%(particle:I:|manner:Dh:|manner:Dh:|time:Dh:),
        'delim': '',
    },
]
```

---

item_class
alias of *ckipnlp.container.parse.ParseClause*

class ckipnlp.container.parse.**ParseParagraph**(*initlist=None*)
Bases: *ckipnlp.container.base.BaseList*

A list of parse sentence.

Data Structure Examples

Text format Used for `to_text()`.

```
[ # delim are ignored
    [ # Sentence 1
        'S(Head:Nab:|particle:Td:)',
        '%(particle:I:|manner:Dh:|manner:Dh:|time:Dh:),
    ],
```

(continues on next page)

(continued from previous page)

```
    [ # Sentence 2
        None,
        'VP(Head:VH11:|particle:Ta:),
        'S(agent:NP(apposition:Nba:|Head:Nhaa:)|Head:VE2:)',
    ],
]
```

List format Used for `from_list()`, and `to_list()`.

```
[
    [ # Sentence 1
        [
            'S(Head:Nab:|particle:Td:)',
            '',
        ],
        [
            '%(particle:I:|manner:Dh:|manner:Dh:|time:Dh:),
            '',
        ],
    ],
    [ # Sentence 2
        [
            None,
            '',
        ],
        [
            'VP(Head:VH11:|particle:Ta:),
            '',
        ],
        [
            'S(agent:NP(apposition:Nba:|Head:Nhaa:)|Head:VE2:)',
            '',
        ],
    ],
]
```

Dict format Used for `from_dict()`, and `to_dict()`.

```
[
    [ # Sentence 1
        {
            'clause': 'S(Head:Nab:|particle:Td:)',
            'delim': '',
        },
        {
            'clause': '%(particle:I:|manner:Dh:|manner:Dh:|time:Dh:),
            'delim': '',
        },
    ],
    [ # Sentence 2
        {
            'clause': None,
            'delim': '',
        },
        {
            'clause': 'VP(Head:VH11:|particle:Ta:),
            'delim': '',
```

(continues on next page)

(continued from previous page)

```
        },
        {
            'clause': 'S(agent:NP(apposition:Nba:|Head:Nhaa:)|Head:VE2:)',
            'delim': '',
        },
    ],
]
```

item_class
alias of *ckipnlp.container.parse.ParseSentence*

4.1.6 ckipnlp.container.seg module

This module provides containers for word-segmented sentences.

class ckipnlp.container.seg.**SegSentence**(*initlist=None*)
Bases: *ckipnlp.container.base.BaseSentence0*

A word-segmented sentence.

Data Structure Examples

Text format Used for `from_text()` and `to_text()`.

```
'' # Words segmented by \u3000 (full-width space)
```

List/Dict format Used for `from_list()`, `to_list()`, `from_dict()`, and `to_dict()`.

```
[ '', '', '', '', '', '', '', ]
```

Note: This class is also used for part-of-speech tagging.

item_class
alias of `str`

class ckipnlp.container.seg.**SegParagraph**(*initlist=None*)
Bases: *ckipnlp.container.base.BaseList*

A list of word-segmented sentences.

Data Structure Examples

Text format Used for `from_text()` and `to_text()`.

```
[
    '',         # Sentence 1
    '', # Sentence 2
]
```

List/Dict format Used for `from_list()`, `to_list()`, `from_dict()`, and `to_dict()`.

```
[
    [ '', '', '', '', '', '', '', ],            # Sentence 1
    [ '', '', '', '', '', '', '', '', '', ], # Sentence 2
]
```

Note: This class is also used for part-of-speech tagging.

---

item_class
alias of *ckipnlp.container.seg.SegSentence*

4.1.7 ckipnlp.container.text module

This module provides containers for text sentences.

class ckipnlp.container.text.**TextParagraph**(*initlist=None*)
Bases: *ckipnlp.container.base.BaseList0*

A list of text sentence.

Data Structure Examples

Text/List/Dict format Used for `from_text()`, `to_text()`, `from_list()`, `to_list()`, `from_dict()`, and `to_dict()`.

```
[
    '',    # Sentence 1
    '', # Sentence 2
]
```

---

item_class
alias of `str`

4.2 ckipnlp.driver package

This module implements CKIPNLP drivers.

Submodules

4.2.1 ckipnlp.driver.base module

This module provides base drivers.

class ckipnlp.driver.base.**DriverRegister**
Bases: `object`

The driver registering utility.

class ckipnlp.driver.base.**BaseDriver**(*, *lazy=False*)
Bases: `object`

The base CKIPNLP driver.

class ckipnlp.driver.base.**DummyDriver**(*, *lazy=False*)
Bases: *ckipnlp.driver.base.BaseDriver*

The dummy driver.

4.2.2 ckipnlp.driver.classic module

This module provides drivers with CkipClassic backend.

class ckipnlp.driver.classic.**CkipClassicWordSegmenter**(*, *lazy=False*, *do_pos=False*, *lexicons=None*)
Bases: *ckipnlp.driver.base.BaseDriver*

The CKIP word segmentation driver with CkipClassic backend.

Parameters

- **lazy** (*bool*) – Lazy initialize the driver.
- **do_pos** (*bool*) – Returns POS-tag or not
- **lexicons** (*Iterable[Tuple[str, str]]*) – A list of the lexicon words and their POS-tags.

__call__(*, *text*)
Apply word segmentation.

Parameters **text** (*TextParagraph*) — The sentences.

Returns

- **ws** (*TextParagraph*) — The word-segmented sentences.
- **pos** (*TextParagraph*) — The part-of-speech sentences. (returns if **do_pos** is set.)

class ckipnlp.driver.classic.**CkipClassicConParser**(*, *lazy=False*)
Bases: ckipnlp.driver.classic._CkipClassicConParser

The CKIP constituency parsing driver with CkipClassic backend.

Parameters **lazy** (*bool*) – Lazy initialize the driver.

__call__(*, *ws*, *pos*)
Apply constituency parsing.

Parameters

- **ws** (*TextParagraph*) — The word-segmented sentences.
- **pos** (*TextParagraph*) — The part-of-speech sentences.

Returns **conparse** (*ParseSentence*) — The constituency-parsing sentences.

class ckipnlp.driver.classic.**CkipClassicConParserClient**(*, *lazy=False*, ***opts*)
Bases: ckipnlp.driver.classic._CkipClassicConParser

The CKIP constituency parsing driver with CkipClassic client backend.

Parameters

- **lazy** (*bool*) – Lazy initialize the driver.
- **username** (*string*) – (*optional*) The username of CkipClassicParserClient.
- **password** (*string*) – (*optional*) The password of CkipClassicParserClient.

Notes

Please register an account at http://parser.iis.sinica.edu.tw/v1/reg.exe and set the environment variables `$CKIPPARSER_USERNAME` and `$CKIPPARSER_PASSWORD`.

__call__ (*, *ws*, *pos*)
Apply constituency parsing.

Parameters

- **ws** (*TextParagraph*) — The word-segmented sentences.
- **pos** (*TextParagraph*) — The part-of-speech sentences.

Returns **conparse** (*ParseSentence*) — The constituency-parsing sentences.

4.2.3 ckipnlp.driver.coref module

This module provides built-in coreference resolution driver.

class ckipnlp.driver.coref.**CkipCorefChunker**(*, *lazy=False*)
Bases: *ckipnlp.driver.base.BaseDriver*

The CKIP coreference resolution driver.

Parameters **lazy** (*bool*) – Lazy initialize the driver.

__call__ (*, *conparse*)
Apply coreference delectation.

Parameters **conparse** (*ParseParagraph*) — The constituency-parsing sentences.

Returns **coref** (*CorefParagraph*) — The coreference results.

static transform_ws (*, *text*, *ws*, *ner*)
Transform word-segmented sentence lists (create a new instance).

static transform_pos (*, *ws*, *pos*, *ner*)
Transform pos-tag sentence lists (modify in-place).

4.2.4 ckipnlp.driver.ss module

This module provides built-in sentence segmentation driver.

class ckipnlp.driver.ss.**CkipSentenceSegmenter**(*, *lazy=False*, *delims='\n'*, *keep_delims=False*)
Bases: *ckipnlp.driver.base.BaseDriver*

The CKIP sentence segmentation driver.

Parameters

- **lazy** (*bool*) – Lazy initialize the driver.
- **delims** (*str*) – The delimiters.
- **keep_delims** (*bool*) – Keep the delimiters.

__call__ (*, *raw*, *keep_all=True*)
Apply sentence segmentation.

Parameters **raw** (*str*) — The raw text.

Returns **text** (*TextParagraph*) — The sentences.

4.2.5 ckipnlp.driver.tagger module

This module provides drivers with CkipTagger backend.

class ckipnlp.driver.tagger.**CkipTaggerWordSegmenter**(*, *lazy=False*, *disable_cuda=True*, *recommend_lexicons={}*, *coerce_lexicons={}*, ***opts*)

Bases: *ckipnlp.driver.base.BaseDriver*

The CKIP word segmentation driver with CkipTagger backend.

Parameters

- **lazy** (*bool*) – Lazy initialize the driver.
- **disable_cuda** (*bool*) – Disable GPU usage.
- **recommend_lexicons** (*Mapping[str, float]*) – A mapping of lexicon words to their relative weights.
- **coerce_lexicons** (*Mapping[str, float]*) – A mapping of lexicon words to their relative weights.

Other Parameters ****opts** – Extra options for `ckiptagger.WS.__call__()`. (Please refer https://github.com/ckiplab/ckiptagger#4-run-the-ws-pos-ner-pipeline for details.)

__call__(*, *text*)

Apply word segmentation.

Parameters **text** (*TextParagraph*) — The sentences.

Returns **ws** (*TextParagraph*) — The word-segmented sentences.

class ckipnlp.driver.tagger.**CkipTaggerPosTagger**(*, *lazy=False*, *disable_cuda=True*, ***opts*)

Bases: *ckipnlp.driver.base.BaseDriver*

The CKIP part-of-speech tagging driver with CkipTagger backend.

Parameters

- **lazy** (*bool*) – Lazy initialize the driver.
- **disable_cuda** (*bool*) – Disable GPU usage.

Other Parameters ****opts** – Extra options for `ckiptagger.POS.__call__()`. (Please refer https://github.com/ckiplab/ckiptagger#4-run-the-ws-pos-ner-pipeline for details.)

__call__(*, *text*)

Apply part-of-speech tagging.

Parameters **ws** (*TextParagraph*) — The word-segmented sentences.

Returns **pos** (*TextParagraph*) — The part-of-speech sentences.

class ckipnlp.driver.tagger.**CkipTaggerNerChunker**(*, *lazy=False*, *disable_cuda=True*, ***opts*)

Bases: *ckipnlp.driver.base.BaseDriver*

The CKIP named-entity recognition driver with CkipTagger backend.

Parameters

- **lazy** (*bool*) – Lazy initialize the driver.
- **disable_cuda** (*bool*) – Disable GPU usage.

Other Parameters ****opts** – Extra options for `ckiptagger.NER.__call__()`. (Please refer https://github.com/ckiplab/ckiptagger#4-run-the-ws-pos-ner-pipeline for details.)

__call__(*, *text*)

Apply named-entity recognition.

Parameters

- **ws** (*TextParagraph*) — The word-segmented sentences.
- **pos** (*TextParagraph*) — The part-of-speech sentences.

Returns **ner** (*NerParagraph*) — The named-entity recognition results.

4.3 ckipnlp.pipeline package

This module implements CKIPNLP pipelines.

Submodules

4.3.1 ckipnlp.pipeline.coref module

This module provides coreference resolution pipeline.

class ckipnlp.pipeline.coref.**CkipCorefDocument**(*, *ws=None*, *pos=None*, *conparse=None*, *coref=None*)

Bases: `collections.abc.Mapping`

The coreference document.

Variables

- **ws** (*SegParagraph*) – The word-segmented sentences.
- **pos** (*SegParagraph*) – The part-of-speech sentences.
- **conparse** (*ParseParagraph*) – The constituency sentences.
- **coref** (*CorefParagraph*) – The coreference resolution results.

class ckipnlp.pipeline.coref.**CkipCorefPipeline**(*, *coref_chunker='default'*, *lazy=True*, *opts={}*, ***kwargs*)

Bases: *ckipnlp.pipeline.kernel.CkipPipeline*

The coreference resolution pipeline.

Parameters

- **sentence_segmenter** (*str*) – The type of sentence segmenter.
- **word_segmenter** (*str*) – The type of word segmenter.
- **pos_tagger** (*str*) – The type of part-of-speech tagger.
- **ner_chunker** (*str*) – The type of named-entity recognition chunker.
- **con_parser** (*str*) – The type of constituency parser.

- **coref_chunker** (*str*) – The type of coreference resolution chunker.

Other Parameters

- **lazy** (*bool*) – Lazy initialize the drivers.
- **opts** (*Dict[str, Dict]*) – The driver options. Key: driver name (e.g. *'sentence_segmenter'*); Value: a dictionary of options.

__call__(*doc*)

Apply coreference delectation.

Parameters **doc** (*CkipDocument*) – The input document.

Returns **corefdoc** (*CkipCorefDocument*) – The coreference document.

Note: **doc** is also modified if necessary dependencies (**ws**, **pos**, **ner**) is not computed yet.

---

get_coref(*doc*, *corefdoc*)

Apply coreference delectation.

Parameters

- **doc** (*CkipDocument*) – The input document.
- **corefdoc** (*CkipCorefDocument*) – The input document for coreference.

Returns **corefdoc.coref** (*CorefParagraph*) – The coreference results.

Note: This routine modify **corefdoc** inplace.

doc is also modified if necessary dependencies (**ws**, **pos**, **ner**) is not computed yet.

4.3.2 ckipnlp.pipeline.kernel module

This module provides kernel CKIPNLP pipeline.

class ckipnlp.pipeline.kernel.**CkipDocument**(*, *raw=None*, *text=None*, *ws=None*, *pos=None*, *ner=None*, *conparse=None*)

Bases: collections.abc.Mapping

The kernel document.

Variables

- **raw** (*str*) – The unsegmented text input.
- **text** (*TextParagraph*) – The sentences.
- **ws** (*SegParagraph*) – The word-segmented sentences.
- **pos** (*SegParagraph*) – The part-of-speech sentences.
- **ner** (*NerParagraph*) – The named-entity recognition results.
- **conparse** (*ParseParagraph*) – The constituency-parsing sentences.

class ckipnlp.pipeline.kernel.**CkipPipeline**(*, *sentence_segmenter='default'*, *word_segmenter='tagger'*, *pos_tagger='tagger'*, *con_parser='classic-client'*, *ner_chunker='tagger'*, *lazy=True*, *opts={}*)

Bases: object

The kernel pipeline.

Parameters

- **sentence_segmenter** (*str*) – The type of sentence segmenter.
- **word_segmenter** (*str*) – The type of word segmenter.
- **pos_tagger** (*str*) – The type of part-of-speech tagger.
- **ner_chunker** (*str*) – The type of named-entity recognition chunker.
- **con_parser** (*str*) – The type of constituency parser.

Other Parameters

- **lazy** (*bool*) – Lazy initialize the drivers.
- **opts** (*Dict[str, Dict]*) – The driver options. Key: driver name (e.g. *'sentence_segmenter'*); Value: a dictionary of options.

get_text(*doc*)

Apply sentence segmentation.

Parameters **doc** (*CkipDocument*) – The input document.

Returns **doc.text** (*TextParagraph*) – The sentences.

Note: This routine modify **doc** inplace.

get_ws(*doc*)

Apply word segmentation.

Parameters **doc** (*CkipDocument*) – The input document.

Returns **doc.ws** (*SegParagraph*) – The word-segmented sentences.

Note: This routine modify **doc** inplace.

get_pos(*doc*)

Apply part-of-speech tagging.

Parameters **doc** (*CkipDocument*) – The input document.

Returns **doc.pos** (*SegParagraph*) – The part-of-speech sentences.

Note: This routine modify **doc** inplace.

get_ner(*doc*)

Apply named-entity recognition.

Parameters **doc** (*CkipDocument*) – The input document.

Returns **doc.ner** (*NerParagraph*) – The named-entity recognition results.

Note: This routine modify **doc** inplace.

get_conparse(*doc*)
Apply constituency parsing.

Parameters **doc** (*CkipDocument*) – The input document.

Returns **doc.conparse** (*ParseParagraph*) – The constituency parsing sentences.

Note: This routine modify **doc** inplace.

4.4 ckipnlp.util package

This module implements extra utilities for CKIPNLP.

Submodules

4.4.1 ckipnlp.util.data module

This module implements data loading utilities for CKIPNLP.

ckipnlp.util.data.**get_tagger_data**()
Get CkipTagger data directory.

ckipnlp.util.data.**install_tagger_data**(*src_dir*, *, *copy=False*)
Link/Copy CkipTagger data directory.

ckipnlp.util.data.**download_tagger_data**()
Download CkipTagger data directory.

4.4.2 ckipnlp.util.logger module

This module implements logging utilities for CKIPNLP.

ckipnlp.util.logger.**get_logger**()
Get the CKIPNLP logger.

CHAPTER

FIVE

INDEX

CHAPTER

SIX

MODULE INDEX

PYTHON MODULE INDEX

C

INDEX

Symbols

B

C

D

F

G

I

M

N

P

S

T

W