jamdict

Le Tuan Anh

Jun 06, 2021

CONTENTS

Jamdict is a Python 3 library for manipulating Jim Breen's JMdict, KanjiDic2, JMnedict and kanji-radical mappings.

CHAPTER

ONE

WELCOME

Are you new to this documentation? Here are some useful pages:

- Want to try out Jamdict package? Try Jamdict online demo
- Want some useful code samples? See *Common Recipes*.
- Want to look deeper into the package? See *jamdict APIs*.
- If you want to help developing Jamdict, please visit *Contributing* page.

CHAPTER

TWO

MAIN FEATURES

- Support querying different Japanese language resources
  - Japanese-English dictionary JMDict
  - Kanji dictionary KanjiDic2
  - Kanji-radical and radical-kanji maps KRADFILE/RADKFILE
  - Japanese Proper Names Dictionary (JMnedict)
- Fast look up (dictionaries are stored in SQLite databases)
- Command-line lookup tool *(Example)*

Contributors are welcome! . If you want to help developing Jamdict, please visit *Contributing* page.

CHAPTER

THREE

INSTALLATION

Jamdict and jamdict-data are both available on PyPI and can be installed using pip. For more information please see *Installation* page.

```
pip install jamdict jamdict-data
```

Also, there is an online demo Jamdict virtual machine to try out on Repl.it

https://replit.com/@tuananhle/jamdict-demo

CHAPTER

FOUR

SAMPLE JAMDICT PYTHON CODE

Looking up words

```
>>> from jamdict import Jamdict
>>> jam = Jamdict()
>>> result = jam.lookup('')
>>> for word in result.entries:
...     print(word)
...
[id#1194500]  () : 1. flower/blossom/bloom/petal ((noun (common) (futsuumeishi))) 2.␣
↪cherry blossom 3. beauty 4. blooming (esp. of cherry blossoms) 5. ikebana 6. Japanese␣
↪playing cards 7. (the) best
[id#1486720]  () : nose ((noun (common) (futsuumeishi)))
[id#1581610]  () : 1. end (e.g. of street)/tip/point/edge/margin ((noun (common)␣
↪(futsuumeishi))) 2. beginning/start/first 3. odds and ends/scrap/odd bit/least
[id#1634180]  () : snivel/nasal mucus/snot ((noun (common) (futsuumeishi)))
```

Looking up kanji characters

```
>>> for c in result.chars:
...     print(repr(c))
...
:7:flower
:10:splendor,flower,petal,shine,luster,ostentatious,showy,gay,gorgeous
:14:nose,snout
:14:edge,origin,end,point,border,verge,cape
:9:tear,nasal discharge
```

Looking up named entities

```
>>> result = jam.lookup('%')
>>> for name in result.names:
...     print(name)
...
[id#5053163]  : Disney (family or surname/company name)
[id#5741091]  : Disneyland (place name)
```

See *Common Recipes* for more code samples.

CHAPTER

FIVE

COMMAND LINE TOOLS

Jamdict can be used from the command line.

```
python3 -m jamdict lookup
========================================
Found entries
========================================
Entry: 1264430 | Kj:   | Kn:
--------------------
1. linguistics ((noun (common) (futsuumeishi)))

========================================
Found characters
========================================
Char:  | Strokes: 7
--------------------
Readings: yan2, eon, , Ngôn, Ngân, , , .,
Meanings: say, word
Char:  | Strokes: 14
--------------------
Readings: yu3, yu4, eo, , Ng, Ng, , ., .
Meanings: word, speech, language
Char:  | Strokes: 8
--------------------
Readings: xue2, hag, , Hoc, , .
Meanings: study, learning, science

No name was found.
```

To show help you may use

```
python3 -m jamdict --help
```

CHAPTER

SIX

DOCUMENTATION

6.1 Installation

jamdict and jamdict dictionary data are both available on PyPI and can be installed using *pip*.

```
pip install --user jamdict jamdict-data
# pip script sometimes doesn't work properly
# so you may want to try this instead
python3 -m pip install jamdict jamdict-data
```

Note: When you use `pip install` in a virtual environment, especially the ones created via `python3 -m venv`, wheel support can be missing. `jamdict-data` relies on wheel/pip to extract xz-compressed database and this may cause a problem. If you encounter any error, please make sure that wheel is available

```
# list all available packages in pip
pip list
# ensure wheel support in pip
pip install -U wheel
```

You may need to uninstall `jamdict-data` before reinstalling it.

```
pip uninstall jamdict-data
```

6.1.1 Download database file manually

This should not be useful anymore from version 0.1a8 with the release of the jamdict_data package on PyPI. If for some reason you want to download and install jamdict database by yourself, here are the steps:

1. Download the offical, pre-compiled jamdict database (`jamdict-0.1a7.tar.xz`) from Google Drive https://drive.google.com/drive/u/1/folders/1z4zF9ImZlNeTZZplflvvnpZfJp3WVLPk
2. Extract and copy `jamdict.db` to jamdict data folder (defaulted to `~/.jamdict/data/jamdict.db`)
3. To know where to copy data files you can use *python3 -m jamdict info* command via a terminal:

```
python3 -m jamdict info
# Jamdict 0.1a8
# Python library for manipulating Jim Breen's JMdict, KanjiDic2, KRADFILE and JMnedict
#
```

(continues on next page)

(continued from previous page)

```
# Basic configuration
# ------------------------------------------------------------
# JAMDICT_HOME             : ~/local/jamdict
# jamdict_data availability: False
# Config file location     : /home/tuananh/.jamdict/config.json
#
# Custom Data files
# ------------------------------------------------------------
# Jamdict DB location: ~/local/jamdict/data/jamdict.db - [OK]
# JMDict XML file    : ~/local/jamdict/data/JMdict_e.gz - [OK]
# KanjiDic2 XML file : ~/local/jamdict/data/kanjidic2.xml.gz - [OK]
# JMnedict XML file : ~/local/jamdict/data/JMnedict.xml.gz - [OK]
#
# Others
# ------------------------------------------------------------
# lxml availability: False
```

6.1.2 Build database file from source

Normal users who just want to look up the dictionaries do not have to do this. If you are a developer and want to build jamdict database from source, copy the dictionary source files to jamdict data folder. The original XML files can be downloaded either from the official website https://www.edrdg.org/ or from this jamdict Google Drive folder.

To find out where to copy the files or whether they are recognised by jamdict, you may use the command *python3 -m jamdict info* as in the section above.

You should make sure that all files under the section *Custom data files* are all marked [OK]. After that you should be able to build the database with the command:

```
python3 -m jamdict import
```

Note on XML parser: jamdict will use *lxml* instead of Python 3 default *xml* when it is available.

6.2 Tutorials

6.2.1 Getting started

Just install `jamdict` and `jamdict_data` packages via pip and you are ready to go.

```
from jamdict import Jamdict
jam = Jamdict()
```

The most useful function is *jamdict.util.Jamdict.lookup()*. For example:

```
# use wildcard matching to find any word, or Kanji character, or name
# that starts with  and ends with
result = jam.lookup('%')
```

To access the result object you may use:

```
# print all word entries
for entry in result.entries:
     print(entry)

# [id#1358280]  () : 1. to eat ((Ichidan verb|transitive verb)) 2. to live on (e.g. a␣
↪salary)/to live off/to subsist on
# [id#1358300]  () : to overeat ((Ichidan verb|transitive verb))
# [id#1852290]  () : to be used to eating ((Ichidan verb|transitive verb))
# [id#2145280]  () : to start eating ((Ichidan verb))
# [id#2449430]  () : to start eating ((Ichidan verb))
# [id#2671010]  () : to be used to eating/to become used to eating/to be accustomed to␣
↪eating/to acquire a taste for ((Ichidan verb))
# [id#2765050]  () : 1. to be able to eat ((Ichidan verb|intransitive verb)) 2. to be␣
↪edible/to be good to eat ((pre-noun adjectival (rentaishi)))
# [id#2795790]  () : to taste and compare several dishes (or foods) of the same type␣
↪((Ichidan verb|transitive verb))
# [id#2807470]  () : to eat together (various foods) ((Ichidan verb))

# print all related characters
for c in result.chars:
    print(repr(c))

# :9:eat,food
# :12:eat,drink,receive (a blow),(kokuji)
# :12:overdo,exceed,go beyond,error
# :5:adhere,attach,refer to,append
# :8:commence,begin
# :11:hang,suspend,depend,arrive at,tax,pour
# :14:accustomed,get used to,become experienced
# :4:compare,race,ratio,Philippines
# :6:fit,suit,join,0.1
```

6.3 Common Recipes

Warning: THIS SECTION IS STILL UNDER CONSTRUCTION

All code here assumed that you have created a Jamdict object named `jam`, like this

```
>>> from jamdict import Jamdict
>>> jam = Jamdict()
```

6.3.1 High-performance tuning

When you need to do a lot of queries on the database, it is possible to load the whole database into memory to boost up querying performance (This will takes about 400 MB of RAM) by using the *memory_mode* keyword argument, like this:

```
>>> from jamdict import Jamdict
>>> jam = Jamdict(memory_mode=True)
```

The first query will be extremely slow (it may take about a minute for the whole database to be loaded into memory) but subsequent queries will be much faster.

6.3.2 Iteration search

Sometimes people want to look through a set of search results only once and determine which items to keep and then discard the rest. In these cases *lookup_iter* should be used. This function returns an *IterLookupResult* object immediately after called. Users may loop through `result.entries`, `result.chars`, and `result.names` exact one loop for each set to find the items that they want. Users will have to store the desired word entries, characters, and names by themselves since they are discarded after yield.

```
>>> res = jam.lookup_iter("")
>>> for word in res.entries:
...     print(word)  # do somethign with the word
>>> for c in res.chars:
...     print(c)
>>> for name in res.names:
...     print(name)
```

6.3.3 Part-of-speeches and named-entity types

Use *Jamdict.all_pos* to list all available part-of-speeches and *Jamdict.all_ne_type* named-entity types:

```
>>> for pos in jam.all_pos():
...     print(pos)  # pos is a string
>>> for ne_type in jam.all_ne_type():
...     print(ne_type)  # ne_type is a string
```

To filter words by part-of-speech use the keyword argument `pos` in *loookup()* or *lookup_iter()* functions.

For example to look for all “” that are nouns use:

```
>>> result = jam.lookup("", pos=["noun (common) (futsuumeishi)"])
```

To search for all named-entities that are “surname” use:

```
>>> result = jam.lookup("surname")
```

6.3.4 Kanjis and radical/components (KRAD/RADK mappings)

Jamdict has built-in support for KRAD/RADK (i.e. kanji-radical and radical-kanji mapping). The terminology of radicals/components used by Jamdict can be different from else where.

- A radical in Jamdict is a principal component, each character has only one radical.
- A character may be decomposed into several writing components.

By default jamdict provides two maps:

- jam.krad is a Python dict that maps characters to list of components.
- jam.radk is a Python dict that maps each available components to a list of characters.

```
# Find all writing components (often called "radicals") of the character
print(jam.krad[''])
# ['', '', '', '']

# Find all characters with the component
chars = jam.radk['']
print(chars)
# {'', '', '', '', ''}

# look up the characters info
result = jam.lookup(''.join(chars))
for c in result.chars:
    print(c, c.meanings())
#  ['cover of tripod cauldron']
#  ['large tripod cauldron with small']
#  ['incense tripod']
#  ['three legged kettle']
#  []
```

6.3.5 Finding name entities

```
# Find all names that contain the string
result = jam.lookup('%%')
for name in result.names:
    print(name)

# [id#5025685]  () : Kyu-ti- Suzuki (1969.10-) (full name of a particular person)
# [id#5064867]  () : Papaiya Suzuki (full name of a particular person)
# [id#5089076]  () : Rajikaru Suzuki (full name of a particular person)
# [id#5259356]  () : Kitsunezakisuzukihinata (place name)
# [id#5379158]  () : Kosuzuki (family or surname)
# [id#5398812]  () : Kamisuzuki (family or surname)
# [id#5465787]  () : Kawasuzuki (family or surname)
# [id#5499409]  () : Oosuzuki (family or surname)
# [id#5711308]  () : Susuki (family or surname)
# ...
```

6.3.6 Exact matching

Use exact matching for faster search

```
# Find an entry (word, name entity) by idseq
result = jam.lookup('id#5711308')
print(result.names[0])
# [id#5711308]  () : Susuki (family or surname)
result = jam.lookup('id#1467640')
print(result.entries[0])
#  () : 1. cat 2. shamisen 3. geisha 4. wheelbarrow 5. clay bed-warmer 6. bottom/
↪submissive partner of a homosexual relationship

# use exact matching to increase searching speed (thanks to @reem-codes)
result = jam.lookup('')

for entry in result.entries:
    print(entry)

# [id#1467640]  () : 1. cat ((noun (common) (futsuumeishi))) 2. shamisen 3. geisha 4.␣
↪wheelbarrow 5. clay bed-warmer 6. bottom/submissive partner of a homosexual␣
↪relationship
# [id#2698030]  () : cat ((noun (common) (futsuumeishi)))
```

6.3.7 Low-level data queries

It's possible to access to the dictionary data by querying database directly using lower level APIs. However these are prone to future changes so please keep that in mind.

When you create a Jamdict object, you have direct access to the underlying databases, via these properties

```
from jamdict import Jamdict
jam = Jamdict()
>>> jam.jmdict    # jamdict.JMDictSQLite object for accessing word dictionary
>>> jam.kd2       # jamdict.KanjiDic2SQLite object, for accessing kanji dictionary
>>> jam.jmnedict  # jamdict.JMNEDictSQLite object, for accessing named-entities␣
↪dictionary
```

You can perform database queries on each of these databases by obtaining a database cursor with `ctx()` function (i.e. database query context).

For example the following code list down all existing part-of-speeches in the database.

```
# returns a list of sqlite3.Row object
pos_rows = jam.jmdict.ctx().select("SELECT DISTINCT text FROM pos")

# access columns in each query row by name
all_pos = [x['text'] for x in pos_rows]

# sort all POS
all_pos.sort()
for pos in all_pos:
    print(pos)
```

For more information, please see Jamdict database schema.

Say we want to get all irregular suru verbs, we can start with finding all Sense IDs with pos = `suru verb - irregular`, and then find all the Entry idseq connected to those Senses.

Words (and also named entities) can be retrieved directly using their `idseq`. Each word may have many Senses (meaning) and each Sense may have different pos.

```
# Entry (idseq) --(has many)--> Sense --(has many)--> pos
```

Note: Tips: Since we hit the database so many times (to find the IDs, to retrieve each word, etc.), we also should consider to reuse the database connection using database context to have better performance (`with jam.jmdict.ctx() as ctx:` and `ctx=ctx` in the code below).

Here is the sample code:

```
# find all idseq of lexical entry (i.e. words) that have at least 1 sense with pos =␣
↪suru verb - irregular
with jam.jmdict.ctx() as ctx:
    # query all word's idseqs
    rows = ctx.select(
        query="SELECT DISTINCT idseq FROM Sense WHERE ID IN (SELECT sid FROM pos WHERE␣
↪text = ?) LIMIT 10000",
        params=("suru verb - irregular",))
    for row in rows:
        # reuse database connection with ctx=ctx for better performance
        word = jam.jmdict.get_entry(idseq=row['idseq'], ctx=ctx)
        print(word)
```

6.4 jamdict APIs

An overview of jamdict modules.

Warning: THIS SECTION IS STILL UNDER CONSTRUCTION Help is much needed.

class jamdict.util.**Jamdict**(*db_file=None*, *kd2_file=None*, *jmd_xml_file=None*, *kd2_xml_file=None*, *auto_config=True*, *auto_expand=True*, *reuse_ctx=True*, *jmnedict_file=None*, *jmnedict_xml_file=None*, *memory_mode=False*, ***kwargs*)

Main entry point to access all available dictionaries in jamdict.

```
>>> from jamdict import Jamdict
>>> jam = Jamdict()
>>> result = jam.lookup('%')
# print all word entries
>>> for entry in result.entries:
>>>     print(entry)
# print all related characters
>>> for c in result.chars:
>>>     print(repr(c))
```

To filter results by `pos`, for example look for all “” that are nouns, use:

```
>>> result = jam.lookup("", pos=["noun (common) (futsuumeishi)"])
```

To search for named-entities by type, use the type string as query. For example to search for all "surname" use:

```
>>> result = jam.lookup("surname")
```

To find out which part-of-speeches or named-entities types are available in the dictionary, use *Jamdict.all_pos* and *Jamdict.all_ne_type*.

Jamdict >= 0.1a10 support `memory_mode` keyword argument for reading the whole database into memory before querying to boost up search speed. The database may take about a minute to load. Here is the sample code:

```
>>> jam = Jamdict(memory_mode=True)
```

When there is no suitable database available, Jamdict will try to use database from jamdict-data package by default. If there is a custom database available in configuration file, Jamdict will prioritise to use it over the `jamdict-data` package.

all_ne_type(*ctx=None*) → List[str]
Find all available named-entity types

Returns A list of named-entity types (a list of strings)

all_pos(*ctx=None*) → List[str]
Find all available part-of-speeches

Returns A list of part-of-speeches (a list of strings)

lookup(*query*, *strict_lookup=False*, *lookup_chars=True*, *ctx=None*, *lookup_ne=True*, *pos=None*, ***kwargs*) → *jamdict.util.LookupResult*
Search words, characters, and characters.

Keyword arguments:

Parameters

- **query** – Text to query, may contains wildcard characters. Use *?* for 1 exact character and % to match any number of characters.
- **strict_lookup** (*bool*) – only look up the Kanji characters in query (i.e. discard characters from variants)
- **pos** (*list of strings*) – Filter words by part-of-speeches
- **ctx** – database access context, can be reused for better performance. Normally users do not have to touch this and database connections will be reused by default.
- **lookup_ne** (*bool*) – set lookup_ne to False to disable name-entities lookup

Param lookup_chars: set lookup_chars to False to disable character lookup

Returns Return a LookupResult object.

Return type *jamdict.util.LookupResult*

```
>>> # match any word that starts with "" and ends with "" (anything from␣
↪between is fine)
>>> jam = Jamdict()
>>> results = jam.lookup('%')
```

lookup_iter(*query*, *strict_lookup=False*, *lookup_chars=True*, *lookup_ne=True*, *ctx=None*, *pos=None*, ***kwargs*) → *jamdict.util.LookupResult*

Search for words, characters, and characters iteratively.

An *IterLookupResult* object will be returned instead of the normal `LookupResult`. `res.entries`, `res.chars`, `res.names` are iterators instead of lists and each of them can only be looped through once. Users have to store the results manually.

```
>>> res = jam.lookup_iter("")
>>> for word in res.entries:
...     print(word)  # do somethign with the word
>>> for c in res.chars:
...     print(c)
>>> for name in res.names:
...     print(name)
```

Keyword arguments:

Parameters

- **query** – Text to query, may contains wildcard characters. Use *?* for 1 exact character and *%* to match any number of characters.
- **strict_lookup** (*bool*) – only look up the Kanji characters in query (i.e. discard characters from variants)
- **pos** (*list of strings*) – Filter words by part-of-speeches
- **ctx** – database access context, can be reused for better performance. Normally users do not have to touch this and database connections will be reused by default.
- **lookup_ne** (*bool*) – set lookup_ne to False to disable name-entities lookup

Param lookup_chars: set lookup_chars to False to disable character lookup

Returns Return an IterLookupResult object.

Return type *jamdict.util.IterLookupResult*

property krad

Break a kanji down to writing components

```
>>> jam = Jamdict()
>>> print(jam.krad[''])
['', '', '', '']
```

property memory_mode

if memory_mode = True, Jamdict DB will be loaded into RAM before querying for better performance

property radk

Find all kanji with a writing component

```
>>> jam = Jamdict()
>>> print(jam.radk[''])
{'', '', '', '', ''}
```

property ready: bool

Check if Jamdict database is available

class jamdict.util.**LookupResult**(*entries*, *chars*, *names=None*)

Contain lookup results (words, Kanji characters, or named entities) from Jamdict.

A typical jamdict lookup is like this:

```
>>> jam = Jamdict()
>>> result = jam.lookup('%')
```

The command above returns a *LookupResult* object which contains found words (*entries*), kanji characters (*chars*), and named entities (*names*).

text(*compact=True*, *entry_sep=''*, *separator=' | '*, *no_id=False*, *with_chars=True*) → str
Generate a text string that contains all found words, characters, and named entities.

Parameters

- **compact** – Make the output string more compact (fewer info, fewer whitespaces, etc.)
- **no_id** – Do not include jamdict's internal object IDs (for direct query via API)
- **entry_sep** – The text to separate entries
- **with_chars** – Include characters information

Returns A formatted string ready for display

property chars: Sequence[jamdict.kanjidic2.Character]
A list of found kanji characters

Returns a list of *Character* object

Return type Sequence[*Character*]

property entries: Sequence[jamdict.jmdict.JMDEntry]
A list of words entries

Returns a list of *JMDEntry* object

Return type List[*JMDEntry*]

property names: Sequence[jamdict.jmdict.JMDEntry]
A list of found named entities

Returns a list of *JMDEntry* object

Return type Sequence[*JMDEntry*]

class jamdict.util.**IterLookupResult**(*entries*, *chars=None*, *names=None*)
Contain lookup results (words, Kanji characters, or named entities) from Jamdict.

A typical jamdict lookup is like this:

```
>>> res = jam.lookup_iter("")
```

`res` is an *IterLookupResult* object which contains iterators to scan through found words (`entries`), kanji characters (`chars`), and named entities (*names*) one by one.

```
>>> for word in res.entries:
...     print(word)  # do somethign with the word
>>> for c in res.chars:
...     print(c)
>>> for name in res.names:
...     print(name)
```

property chars
Iterator for looping one by one through all found kanji characters, can only be used once

property entries
Iterator for looping one by one through all found entries, can only be used once

property names
Iterator for looping one by one through all found named entities, can only be used once

class jamdict.jmdict.**JMDEntry**(*idseq=''*)
Represents a dictionary Word entry.

Entries consist of kanji elements, reading elements, general information and sense elements. Each entry must have at least one reading element and one sense element. Others are optional.

XML DTD <!ELEMENT entry (ent_seq, k_ele*, r_ele+, info?, sense+)>

class jamdict.kanjidic2.**Character**
Represent a kanji character.

<!ELEMENT character (literal,codepoint, radical, misc, dic_number?, query_code?, reading_meaning?)*>

property components
Kanji writing components that compose this character

meanings(*english_only=False*)
Accumulate all meanings as a list of string. Each string is a meaning (i.e. sense)

jamdict.krad is a module for retrieving kanji components (i.e. radicals)

6.5 Contributing

There are many ways to contribute to the Jamdict project. The one that Jamdict development team are focusing on at the moment are:

- Fixing *existing bugs*
- Improving query functions
- Improving *documentation*
- Keeping jamdict database up to date

If you have some suggestions or bug reports, please share on jamdict issues tracker.

6.5.1 Fixing bugs

If you found a bug please report at https://github.com/neocl/jamdict/issues

When it is possible, please also share how to reproduce the bugs and a snapshot of jamdict info to help with the bug finding process.

```
python3 -m jamdict info
```

Pull requests are welcome.

6.5.2 Updating Documentation

1. Fork jamdict repository to your own Github account.
2. Clone *jamdict* repository to your local machine.

```
git clone https://github.com/<your-account-name>/jamdict
```

3. Create a virtual environment (optional, but highly recommended)

```
# if you use virtualenvwrapper
mkvirtualenv jamdev
workon jamdev

# if you use Python venv
python3 -m venv .env
. .env/bin/activate
python3 -m pip install --upgrade pip wheel Sphinx
```

4. Build the docs

```
cd jamdict/docs
# compile the docs
make dirhtml
# serve the docs using Python3 built-in development server
# Note: this requires Python >= 3.7 to support --directory
python3 -m http.server 7000 --directory _build/dirhtml
# if you use earlier Python 3, you may use
cd _build/dirhtml
python3 -m http.server 7000
```

5. Now the docs should be ready to view at http://localhost:7000 . You can visit that URL on your browser to view the docs.
6. More information:
   - Sphinx tutorial: https://sphinx-tutorial.readthedocs.io/start/
   - Using *virtualenv*: https://virtualenvwrapper.readthedocs.io/en/latest/install.html
   - Using *venv*: https://docs.python.org/3/library/venv.html

6.5.3 Development

Development contributions are welcome. Setting up development environment for Jamdict should be similar to *Updating Documentation*.

Please contact the development team if you need more information: https://github.com/neocl/jamdict/issues

6.6 Jamdict Changelog

6.6.1 jamdict 0.1a11

- 2021-05-25
- Added `lookup_iter()` for iteration search
- Added `pos` filter for filtering words by part-of-speeches
- Added `all_pos()` and `all_ne_type()` to Jamdict to list part-of-speeches and named-entity types
- Better version checking in `__version__.py`
- Improved documentation
- 2021-05-29
  - (.post1) Sorted kanji readings to have on & kun readings listed first
  - (.post1) Add `on_readings`, `kun_readings`, and `other_readings` filter to `kanjidic2.RMGroup`

6.6.2 jamdict 0.1a10

- 2021-05-19
- Added `memory_mode` keyword to load database into memory before querying to boost up performance
- Improved import performance by using puchikarui's `buckmode`
- Tested with both puchikarui 0.1.* and 0.2.*

6.6.3 jamdict 0.1a9

- 2021-04-19
- Fix data audit query
- Enhanced `Jamdict()` constructor. `Jamdict('/path/to/jamdict.db')` works properly.
- Code quality review
- Automated documentation build via readthedocs.org

6.6.4 jamdict 0.1a8

- 2021-04-15
- Make `lxml` optional
- Data package can be installed via PyPI with `jamdict_data` package
- Make configuration file optional as data files can be installed via PyPI.

6.6.5 jamdict 0.1a7

- 2020-05-31
- Added Japanese Proper Names Dictionary (JMnedict) support
- Included built-in KRADFILE/RADKFile support
- Improved command line tools (json, compact mode, etc.)

6.6.6 Older versions

- 2017-08-18
  - Support KanjiDic2 (XML/SQLite formats)
- 2016-11-09
  - Release first version to Github

CHAPTER

SEVEN

OTHER INFO

7.1 Release Notes

Release notes is available *here*.

7.2 Contributors

- Le Tuan Anh (Maintainer)
- alt-romes
- Matteo Fumagalli
- Reem Alghamdi
- Techno-coder

7.3 Useful links

- jamdict on PyPI: https://pypi.org/project/jamdict/
- jamdict source code: https://github.com/neocl/jamdict/
- Documentation: https://jamdict.readthedocs.io/
- **Dictionaries**
  - JMdict: http://edrdg.org/jmdict/edict_doc.html
  - kanjidic2: https://www.edrdg.org/wiki/index.php/KANJIDIC_Project
  - JMnedict: https://www.edrdg.org/enamdict/enamdict_doc.html
  - KRADFILE: http://www.edrdg.org/krad/kradinf.html

7.4 Indices and tables

- genindex
- modindex
- search

PYTHON MODULE INDEX

j

INDEX