qwikidata

Release 0.4.2

Kensho Technologies LLC

Nov 09, 2022

CONTENTS

CHAPTER

ONE

WELCOME

qwikidata is a Python package with tools that allow you to interact with Wikidata.

The package defines a set of classes that allow you to represent Wikidata entities in a Pythonic way. It also provides a Pythonic way to access three data sources,

- linked data interface
- sparql query service
- json dump

1.1 Quick Install

1.1.1 Requirements

- python >= 3.5

1.1.2 Install with pip

You can install the most recent version using pip,

```
pip install qwikidata
```

1.2 Quick Examples

1.2.1 Linked Data Interface

```
from qwikidata.entity import WikidataItem, WikidataLexeme, WikidataProperty
from qwikidata.linked_data_interface import get_entity_dict_from_api

# create an item representing "Douglas Adams"
Q_DOUGLAS_ADAMS = "Q42"
q42_dict = get_entity_dict_from_api(Q_DOUGLAS_ADAMS)
q42 = WikidataItem(q42_dict)
```

(continues on next page)

(continued from previous page)

```
# create a property representing "subclass of"
P_SUBCLASS_OF = "P279"
p279_dict = get_entity_dict_from_api(P_SUBCLASS_OF)
p279 = WikidataProperty(p279_dict)

# create a lexeme representing "bank"
L_BANK = "L3354"
l3354_dict = get_entity_dict_from_api(L_BANK)
l3354 = WikidataLexeme(l3354_dict)
```

1.2.2 SPARQL Query Service

```
from qwikidata.sparql import (get_subclasses_of_item,
                              return_sparql_query_results)

# send any sparql query to the wikidata query service and get full result back
# here we use an example that counts the number of humans
sparql_query = """
SELECT (COUNT(?item) AS ?count)
WHERE {
        ?item wdt:P31/wdt:P279* wd:Q5 .
}
"""
res = return_sparql_query_results(sparql_query)


# use convenience function to get subclasses of an item as a list of item ids
Q_RIVER = "Q4022"
subclasses_of_river = get_subclasses_of_item(Q_RIVER)
```

1.2.3 JSON Dump

```
import time

from qwikidata.entity import WikidataItem
from qwikidata.json_dump import WikidataJsonDump
from qwikidata.utils import dump_entities_to_json

P_OCCUPATION = "P106"
Q_POLITICIAN = "Q82955"


def has_occupation_politician(item: WikidataItem, truthy: bool = True) -> bool:
    """Return True if the Wikidata Item has occupation politician."""
    if truthy:
        claim_group = item.get_truthy_claim_group(P_OCCUPATION)
    else:
        claim_group = item.get_claim_group(P_OCCUPATION)
```

(continues on next page)

(continued from previous page)

```
    occupation_qids = [
        claim.mainsnak.datavalue.value["id"]
        for claim in claim_group
        if claim.mainsnak.snaktype == "value"
    ]
    return Q_POLITICIAN in occupation_qids


# create an instance of WikidataJsonDump
wjd_dump_path = "wikidata-20190401-all.json.bz2"
wjd = WikidataJsonDump(wjd_dump_path)

# create an iterable of WikidataItem representing politicians
politicians = []
t1 = time.time()
for ii, entity_dict in enumerate(wjd):

    if entity_dict["type"] == "item":
        entity = WikidataItem(entity_dict)
        if has_occupation_politician(entity):
            politicians.append(entity)

    if ii % 1000 == 0:
        t2 = time.time()
        dt = t2 - t1
        print(
            "found {} politicians among {} entities [entities/s: {:.2f}]".format(
                len(politicians), ii, ii / dt
            )
        )

    if ii > 10000:
        break

# write the iterable of WikidataItem to disk as JSON
out_fname = "filtered_entities.json"
dump_entities_to_json(politicians, out_fname)
wjd_filtered = WikidataJsonDump(out_fname)

# load filtered entities and create instances of WikidataItem
for ii, entity_dict in enumerate(wjd_filtered):
    item = WikidataItem(entity_dict)
```

1.3 License

1.4 Copyright

1.5 Important Links

readthedocs | PyPI | github

CHAPTER

TWO

WIKIDATA

This section describes the raw Wikidata data products.

> "Wikidata is a free and open knowledge base that can be read and edited by both humans and machines. Wikidata acts as central storage for the structured data of its Wikimedia sister projects including Wikipedia, Wikivoyage, Wikisource, and others. Wikidata also provides support to many other sites and services beyond just Wikimedia projects! The content of Wikidata is available under a free license, exported using standard formats, and can be interlinked to other open data sets on the linked data web."
>
> —Wikidata

2.1 Linked Data Interface

`qwikidata.linked_data_interface`

2.1.1 Description

Wikidata provides access to knowledge base entity information through a linked data interface,

> "Each item or property has a persistent URI that you obtain by appending its ID (such as Q42 or P12) to the Wikidata concept namespace: http://www.wikidata.org/entity/
>
> For example, the concept URI of Douglas Adams is http://www.wikidata.org/entity/Q42. Note that this URI refers to the real-world person, not Wikidata's description of Douglas Adams. However, it is possible to use the concept URI to access data about Douglas Adams by simply using it as a URL. When you request this URL, it triggers an HTTP redirect that forwards the client to the data URL for Wikidata's data about Douglas Adams: http://www.wikidata.org/wiki/Special:EntityData/Q42. The namespace for Wikidata's data about entities is http://www.wikidata.org/wiki/Special:EntityData/"
>
> —https://www.wikidata.org/wiki/Wikidata:Data_access

2.1.2 Example

```
from qwikidata.linked_data_interface import get_entity_dict_from_api
from qwikidata.entity import WikidataItem, WikidataProperty, WikidataLexeme

q42_dict = get_entity_dict_from_api('Q42')
q42 = WikidataItem(q42_dict)

p279_dict = get_entity_dict_from_api('P279')
```

(continues on next page)

(continued from previous page)

```
p279 = WikidataProperty(p279_dict)

l3_dict = get_entity_dict_from_api('L3')
l3 = WikidataLexeme(l3_dict)
```

2.2 SPARQL End Point

`qwikidata.sparql`

2.2.1 Description

Wikidata provides endpoints to process SPARQL queries,

> SPARQL queries can be submitted directly to the SPARQL endpoint with GET request to *https://query.wikidata.org/bigdata/namespace/wdq/sparql?query={SPARQL}* or the endpoint's alias *https://query.wikidata.org/sparql?query={SPARQL}*. The result is returned as XML by default, or as JSON if either the query parameter format=json or the header Accept: application/sparql-results+json are provided. See the user manual for more detailed information.
>
> —https://www.wikidata.org/wiki/Wikidata:SPARQL_query_service

You can find their GUI implementation here https://query.wikidata.org/

2.2.2 Example

We can find all items that are in the subclass tree of river. Note that,

- `subclass of` is P279
- `river` is Q4022

```
from qwikidata.sparql import return_sparql_query_results

query_string = """
SELECT $WDid
WHERE {
  ?WDid (wdt:P279)* wd:Q4022
}
"""

results = return_sparql_query_results(query_string)
```

Alternatively, we can find all items that have river in their subclass tree.

```
from qwikidata.sparql import return_sparql_query_results

query_string = """
SELECT $WDid
WHERE {
  wd:Q4022 (wdt:P279)* ?WDid
}
```

(continues on next page)

(continued from previous page)

```
"""

results = return_sparql_query_results(query_string)
```

2.3 JSON Dump Files

qwikidata.json_dump

2.3.1 Description

Wikidata provides frequent (every few days) dumps of the knowledge base in the form of compressed JSON files. From the docs,

> "JSON dumps containing all Wikidata entities in a single JSON array can be found under https://dumps.wikimedia.org/wikidatawiki/entities/. The entities in the array are not necessarily in any particular order, e.g., Q2 doesn't necessarily follow Q1. The dumps are being created on a weekly basis.
>
> This is the recommended dump format. Please refer to the JSON structure documentation for information about how Wikidata entities are represented.
>
> Hint: Each entity object (data item or property) is placed on a separate line in the JSON file, so the file can be read line by line, and each line can be decoded separately as an individual JSON object."
>
> —https://www.wikidata.org/wiki/Wikidata:Database_download

2.3.2 Example

```
from qwikidata.json_dump import WikidataJsonDump

wjd = WikidataJsonDump('wikidata-20190107-all.json.bz2')
```

Iteration over the qwikidata.json_dump.WikidataJsonDump object will yield dictionary representations of entities (one entity per iteration).

```
from qwikidata.entity import WikidataItem, WikidataProperty

type_to_entity_class = {"item": WikidataItem, "property": WikidataProperty}
max_entities = 5
entities = []


for ii, entity_dict in enumerate(wjd):
  if ii >= max_entities:
    break
  entity_id = entity_dict["id"]
  entity_type = entity_dict["type"]
  entity = type_to_entity_class[entity_type](entity_dict)
  entities.append(entity)
```

(continues on next page)

(continued from previous page)

```
for entity in entities:
  print(entity)
```

```
WikidataItem(label=Belgium, id=Q31, description=federal constitutional monarchy in␣
↪Western Europe, aliases=['Kingdom of Belgium', 'be', ''], enwiki_title=Belgium)
WikidataItem(label=happiness, id=Q8, description=mental or emotional state of well-being␣
↪characterized by pleasant emotions, aliases=['', ':)', '', 'joy', 'happy'], enwiki_
↪title=Happiness)
WikidataItem(label=George Washington, id=Q23, description=First President of the United␣
↪States, aliases=['Washington', 'President Washington', 'G. Washington', 'Father of the␣
↪United States'], enwiki_title=George Washington)
WikidataItem(label=Jack Bauer, id=Q24, description=character from the television series␣
↪24, aliases=[], enwiki_title=Jack Bauer)
WikidataItem(label=Douglas Adams, id=Q42, description=British author and humorist␣
↪(1952-2001), aliases=['Douglas Noel Adams', 'Douglas Noël Adams', 'Douglas N. Adams'],␣
↪enwiki_title=Douglas Adams)
```

It is also possible to use the `qwikidata.json_dump.WikidataJsonDump.create_chunks()` method to create truncated versions of the json dump file and/or break the original file into chunks,

```
# create a single chunk to get a truncated version of the file
trunc_file_name = wjd.create_chunks(num_lines_per_chunk=5, max_chunks=1)

# or create all the chunks
chunk_file_names = wjd.create_chunks(num_lines_per_chunk=100_000)
```

2.4 Entities

`qwikidata.entity`

2.4.1 Description

The majority of code in this package is dedicated to classes that can be used to represent Wikidata entities. We consider three types of entities,

- `qwikidata.entity.WikidataItem`
- `qwikidata.entity.WikidataProperty`
- `qwikidata.entity.WikidataLexeme`

2.4.2 Examples

In order to use the wikidata entity classes we will need some data. Wikidata makes full dumps of the knowledge base available in JSON format, but we will use their linked data API instead to grab data for just one entity. If you'd like to see the documentation for qwikidata objects that handle either of these things, you can follow the links below,

- `qwikidata.json_dump.WikidataJsonDump`
- `qwikidata.linked_data_interface.get_entity_dict_from_api()`

Note: Currently the JSON dumps provided by Wikidata do not include Lexemes but they are available through the linked data interface. See https://phabricator.wikimedia.org/T195419

Creation

For now, lets just get the raw data dictionary for "Douglas Adams" (aka Q42) and create an instance of `qwikidata.entity.WikidataItem`.

```
>>> from qwikidata.linked_data_interface import get_entity_dict_from_api
>>> from qwikidata.entity import WikidataItem, WikidataProperty

>>> q42_dict = get_entity_dict_from_api("Q42")
>>> q42 = WikidataItem(q42_dict)
```

Basic Data

Instances of this class make basic information about Douglas Adams available via attributes and methods.

```
>>> q42.entity_id
'Q42'

>>> q42.entity_type
'item'

>>> q42.get_label()
'Douglas Adams'

>>> q42.get_description()
'author and humorist'

>>> q42.get_aliases()
['Douglas Noël Adams', 'Douglas Noel Adams', 'Douglas N. Adams']

>>> q42.get_enwiki_title()
'Douglas Adams'

>>> q42.get_sitelinks()["enwiki"]["url"]
'https://en.wikipedia.org/wiki/Douglas_Adams'
```

Note: The `entity_id` and `entity_type` values are singular and come from the top level of the entity dictionary (`q42_dict`) so they are attached to the instance as attributes during initialization. The other data (label, description,

...) is non-singular (has values for many languages) and non-trivial to parse from the entity dictionary. Therefore, we supply this data via methods so that the entity dictionary is only parsed "on demand". This saves a lot of time when iterating over a large number of entities.

In addition, the `__str__` and `__repr__` methods return a summary of this basic info,

```
>>> print(q42)
WikidataItem(label=Douglas Adams, id=Q42, description=author and humorist, aliases=[
↪'Douglas Noël Adams', 'Douglas Noel Adams', 'Douglas N. Adams'], enwiki_title=Douglas␣
↪Adams)
```

By default, these methods return strings in English. Analogous information is available in many different languages by passing the `lang` keyword. For example, the Dutch version of the description of Douglas Adams is,

```
>>> q42.get_description(lang="nl")
'Engelse schrijver (1952-2001)'
```

A list of all the language codes is available from Wikidata.

Claims / Statements

So far we've covered the basic metadata available for an entity (labels, descriptions, aliases, ...). However, the real power of wikidata lies in what are called "claims" or "statements".

> "In Wikidata, a concept, topic, or object is represented by an item. Each item is accorded its own page. A statement is how the information we know about an item—the data we have about it—gets recorded in Wikidata."
>
> —https://www.wikidata.org/wiki/Help:Statements

Lets examine the claims about Douglas Adams with property P69 ("educated at"). Here's what they look like on the Wikidata page,

We can see that there are two claims here, one for "St John's College" (Q691283) and one for "Brentwood School" (Q4961791). The "St John's College" entry has four qualifiers and two references while the "Brentwood School" entry has two qualifiers and zero references.

We can access this data from our Douglas Adams object (*q42*) using the *get_claims* method which returns a dictionary mapping property id to `qwikidata.claim.WikidataClaimGroup`.

```
>>> claim_groups = q42.get_truthy_claim_groups()
>>> p69_claim_group = claim_groups["P69"]
>>> len(p69_claim_group)
2
```

Note: The methods that return claim groups come in "truthy" and "standard" versions,

- `qwikidata.entity.ClaimsMixin.get_claim_group()`
- `qwikidata.entity.ClaimsMixin.get_truthy_claim_group()`
- `qwikidata.entity.ClaimsMixin.get_claim_groups()`
- `qwikidata.entity.ClaimsMixin.get_truthy_claim_groups()`

You almost always want to use the truthy versions. Truthy is defined in the Wikidata RDF dump format docs,

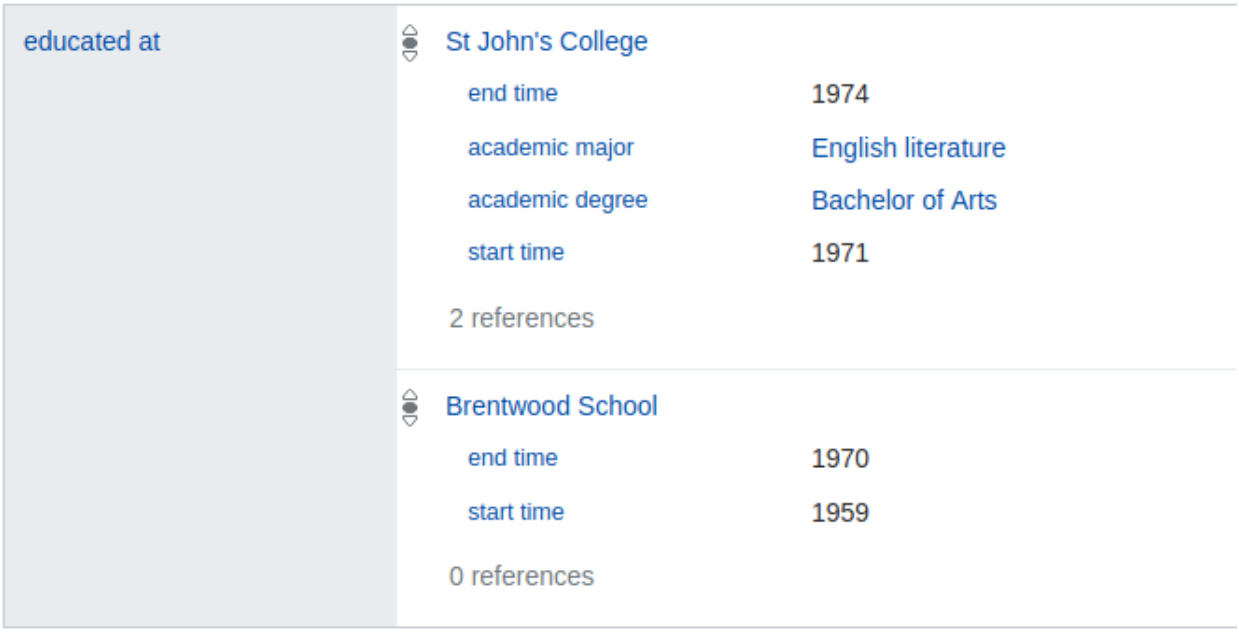

Fig. 1: The P69 ("eduated at") claim group for Q42 ("Douglas Adams") as displayed on the Wikidata webstite (Aug. 2018).

> "Truthy statements represent statements that have the best non-deprecated rank for a given property. Namely, if there is a preferred statement for a property P, then only preferred statements for P will be considered truthy. Otherwise, all normal-rank statements for P are considered truthy."
>
> —RDF dump format docs on truthy statements

Each claim in the claim group has a `main_snak` attribute that represents the primary information of the claim, as well as *qualifiers* and *references* attributes. In this case, the main snak of one claim would reference "St John's College" and the other "Brentwood School". Snaks are a central data structure in Wikidata. They appear in each claim in the following way,

- **main_snak**: An instance of `qwikidata.snak.WikidataSnak`
- **qualifiers** (*OrderedDict*): property id -> list of `qwikidata.claim.WikidataQualifier`
- **references** (*list*): Each element is an instance of `qwikidata.claim.WikidataReference`

Each snak has one datavalue (defined in `qwikidata.datavalue`). The datavalues store the raw data that we are interested in. There are seven basic data types for datavalues and we use classes to represent them,

- `qwikidata.datavalue.GlobeCoordinate`
- `qwikidata.datavalue.MonolingualText`
- `qwikidata.datavalue.Quantity`
- `qwikidata.datavalue.String`
- `qwikidata.datavalue.Time`
- `qwikidata.datavalue.WikibaseEntityId`
- `qwikidata.datavalue.WikibaseUnmappedEntityId`

Now, lets examine the first claim and grab some data.

```
>>> claim = p69_claim_group[0]
>>> print(f"claim.rank={claim.rank}")
claim.rank=normal

>>> qid = claim.mainsnak.datavalue.value["id"]
>>> print(qid)
Q691283

>>> entity = WikidataItem(get_entity_dict_from_api(qid))
>>> print(entity.get_label())
St John's College

>>> for pid, quals in claim.qualifiers.items():
>>>     prop = WikidataProperty(get_entity_dict_from_api(pid))
>>>     for qual in quals:
>>>         if qual.snak.snaktype != "value":
>>>             continue
>>>         else:
>>>             print(f"{prop.get_label()}: {qual.snak.datavalue}")
end time: Time(time=+1974-01-01T00:00:00Z, precision=9)
academic major: WikibaseEntityid(id=Q186579)
academic degree: WikibaseEntityid(id=Q1765120)
start time: Time(time=+1971-00-00T00:00:00Z, precision=9)

>>> for ref_num, ref in enumerate(claim.references):
>>>     print(f"ref num={ref_num}")
>>>     for pid, snaks in ref.snaks.items():
>>>         prop = WikidataProperty(get_entity_dict_from_api(pid))
>>>         for snak in snaks:
>>>             if snak.snaktype != "value":
>>>                 continue
>>>             else:
>>>                 print(f"{prop.get_label()}: {snak.datavalue}")
ref num=0
stated in: WikibaseEntityid(id=Q5375741)
ref num=1
reference URL: String(value=http://www.nndb.com/people/731/000023662/)
language of work or name: WikibaseEntityid(id=Q1860)
publisher: WikibaseEntityid(id=Q1373513)
retrieved: Time(time=+2013-12-07T00:00:00Z, precision=11)
title: MongolingualText(text=Douglas Adams, language=en)
```

Note: There are a few things to note about the code and output above.

- We print the `rank` attribute of the claim. Claims can have three ranks, "preferred", "normal", or "deprecated". You shouldn't have to worry about this is you use "truthy" claims.
- We check the `snaktype` and continue to the next iteration if it is not equal to "value". Snaktypes can be "value", "somevalue", or "novalue". These indicate a known value, an unknown value, and no existing value respectively.
- We have relied on the `__str__` implementations of each datavalue class to present a short string summarizing the information.
- The linked data interface API (i.e. `qwikidata.linked_data_interface.get_entity_dict_from_api()`)

is fine for exploring and prototyping, but for large scale calculations its best to use the JSON dump.

This was just an introduction to get you started. There is lots more to explore. Enjoy!

CHAPTER

THREE

API REFERENCE

CHAPTER

FOUR

PACKAGE SUMMARY

qwikidata.json_dump	Module for Wikidata JSON dumps.
qwikidata.datavalue	Module for Wikidata Datavalues.
qwikidata.snak	Module for Wikidata Snaks.
qwikidata.claim	Module for Wikidata Claims (aka Statements).
qwikidata.entity	Module for Wikidata Entities.
qwikidata.linked_data_interface	Module for Wikidata linked data interface endpoints.
qwikidata.sparql	Module for the Wikidata SPARQL endpint.
qwikidata.typedefs	Module providing Wikidata Types.

4.1 qwikidata.json_dump

Module for Wikidata JSON dumps.

Classes

WikidataJsonDump(filename)	Class for Wikidata JSON dump files.

4.2 qwikidata.datavalue

Module for Wikidata Datavalues.

Functions

get_datavalue_from_snak_dict(snak_dict)	Return a Wikidata Datavalue from a snak dictionary.

Classes

GlobeCoordinate(datavalue_dict)	Class for *globecoordinate* datavalues.
MonolingualText(datavalue_dict)	Class for *monolingualtext* datavalues.
Quantity(datavalue_dict)	Class for *quantity* datavalues.
String(datavalue_dict)	Class for *string* datavalues.
Time(datavalue_dict)	Class for *time* datavalues.
WikibaseEntityId(datavalue_dict)	Class for *wikibase-entityid* datavalues.
WikibaseUnmappedEntityId(datavalue_dict)	Class for *wikibase-unmapped-entityid* datavalues.

4.3 qwikidata.snak

Module for Wikidata Snaks.

Classes

WikidataSnak(snak_dict)	A Wikidata snak.

4.4 qwikidata.claim

Module for Wikidata Claims (aka Statements).

Classes

WikidataClaim(claim_dict)	A claim aka statement about a Wikidata Entity.
WikidataClaimGroup(claim_list)	A sequence of WikidataClaim instances with a common property id.
WikidataQualifier(qualifier_dict)	A qualifier about a claim about a Wikidata Entity.
WikidataReference(reference_dict)	A reference about a claim about a Wikidata Entity.

4.5 qwikidata.entity

Module for Wikidata Entities.

Classes

ClaimsMixin()	Mixin for entities with top level claims (aka statements).
EntityMixin()	Mixin for all entities.
LabelDescriptionAliasMixin()	Mixin for entities with labels, descriptions, and aliases.
WikidataForm(form_dict)	A form associated with a Wikidata Lexeme.
WikidataItem(item_dict)	Class for Wikidata Items.
WikidataLexeme(lexeme_dict)	Class for Wikidata Lexeme.
WikidataProperty(property_dict)	Class for Wikidata Properties.
WikidataSense(sense_dict)	A sense associated with a Wikidata Lexeme.

4.6 qwikidata.linked_data_interface

Module for Wikidata linked data interface endpoints.

Functions

get_entity_dict_from_api(entity_id[, base_url])	Get a dictionary representing a wikidata entity from the linked data interface API.

Exceptions

InvalidEntityId
LdiResponseNotOk

4.7 qwikidata.sparql

Module for the Wikidata SPARQL endpint.

Functions

get_subclasses_of_item(item_id[, return_qids])	Return all subclasses of a wikidata item.
return_sparql_query_results(query_string[, ...])	Send a SPARQL query and return the JSON formatted result.

4.8 qwikidata.typedefs

Module providing Wikidata Types.

Classes

AliasDict(*args, **kwargs)
ClaimDict(*args, **kwargs)
DescriptionDict(*args, **kwargs)
FormDict(*args, **kwargs)
GlobeCoordinateDatavalueDict(*args, **kwargs)
GlobeCoordinateValue(*args, **kwargs)
GlossDict(*args, **kwargs)
ItemDict(*args, **kwargs)
LabelDict(*args, **kwargs)
LemmaDict(*args, **kwargs)
LexemeDict(*args, **kwargs)
MonolingualTextDatavalueDict(*args, **kwargs)
MonolingualTextValue(*args, **kwargs)
PropertyDict(*args, **kwargs)
QualifierDict(*args, **kwargs)
QuantityDatavalueDict(*args, **kwargs)
QuantityValue(*args, **kwargs)
ReferenceDict(*args, **kwargs)
RepresentationDict(*args, **kwargs)
SenseDict(*args, **kwargs)
SitelinkDict(*args, **kwargs)
SnakDict(*args, **kwargs)
StringDatavalueDict(*args, **kwargs)
TimeDatavalueDict(*args, **kwargs)
TimeValue(*args, **kwargs)
WikibaseEntityIdDatavalueDict(*args, **kwargs)
WikibaseEntityIdValue(*args, **kwargs)
WikibaseUnmappedEntityIdDatavalueDict(*args, ...)

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

M

Q