
dabeplech

Release 0.6.0

Kenzo-Hugo Hillion

Mar 26, 2021

DABEPLECH

1 Quickstart	3
Python Module Index	33
Index	35

dabeplech is a modern and light library to perform requests to different bioinformatics APIs with Python 3.6+. The key features of the library are the following:

- **Accessible:** Designed to be easy to use.
- **Open:** Easy to contribute to and add modules to deal with your own API.
- **Standards-based:** Based on OpenAPI, the open standards for APIs.

QUICKSTART

1.1 Installation

```
pip install dabeplich
```

1.2 Example

```
from dabeplich import KEGGAPI

api = KEGGAPI()
kegg_entry = api.get("K00135")

print(kegg_entry.names)
# OUTPUT: ['gabD']
```

1.2.1 Why dabeplich?

The main motivation behind dabeplich is to provide a simple way to query bioinformatics API.

It is also designed in a way to ease any contributions and try to keep its structure as simple as possible.

This library contains models description (thanks to [pydantic](#)). It allows validations of the format and the generation of a documentation of the responses from APIs that are initially not described or following any standards.

1.2.2 Contributing to dabeplich

You have many different ways to contribute to dabeplich:

- Add new API connectors
- [Report a Bug](#)
- Suggesting a new Feature
- Update the Documentation
- Participate to the discussions in the Github issues

Note: For more advanced features, such as the code architecture, we encourage to go through discussions on GitHub before starting any major changes or contributions.

General rules

- *Development environment*: set up your environment to contribute.
- *Contribution Flow*: contribution through GitHub.
- *Code of Conduct*

Adding a new API connector

For the moment, we identify three different scenarios when adding a new API connector:

API already returning JSON

This is the most simple case where you just need to add a new connector:

1. *Adding an API returning JSON*
2. *Completing the Documentation*

API returning different format

1. *Model description of the response*
2. *Parsers*
3. *Adding an API based on a Parser*
4. *Completing the Documentation*

Build API connector from scrapping

1. *Model description of the response*
2. *contrib_scrapper*
3. *Adding an API based on a Scrapper*
4. *Completing the Documentation*

1.2.3 Installation

Requirements

dabeplich works with version of Python $\geq 3.6.0$ and uses the following dependencies:

- [pydantic](#) (==1.5.1)
- [requests](#) (==2.23.0)

Note: We highly recommend the use of a virtual environment such as [virtualenv](#), [pyenv](#) or [conda](#).

Installation procedure

Pip

You can use pip to install dabeplich of the latest stable version published on pypi:

```
pip install dabeplich
```

Manually

Note: This is particularly useful when you wish to install a version under development from any branches of the Github repository.

Clone the repository and install dabeplich with the following commands:

```
git clone https://github.com/khillion/dabeplich.git
cd dabeplich
pip install .
```

Uninstallation procedure

You can remove dabeplich with the following command:

```
pip uninstall dabeplich
```

Note: This will not uninstall dependencies. To do so you can make use of the pip-autoremove tool [pip-autoremove](#) or set up your environment with [pipenv](#) or [poetry](#)...

1.2.4 Basic usage

Get data with API connectors

KEGG example

Get results (as a Pydantic model):

By default, the result is returned using the pydantic model.

```
from dabepilech import KEGGAPI

api = KEGGAPI()
kegg_entry = api.get("K00135")

print(kegg_entry.names)
# OUTPUT: ['gabD']
```

Get results as a Python dict:

If you prefer you can instead get a python dict with the `get_model=False` argument.

```
from dabepilech import KEGGAPI

api = KEGGAPI()
kegg_entry = api.get("K00135", get_model=False)

print(kegg_entry.get('names'))
# OUTPUT: ['gabD']
```

Note: You can also directly retrieve a dict from the model using `.dict()` method.

1.2.5 List of supported APIs

- **KEGG:** Database resource for understanding high-level functions and utilities of the biological system.
- **togoWS:** Uniformed REST API for accessing major bioinformatics data resources and data formats.
- **PDBe:** PDBe is a founding member of the Worldwide Protein Data Bank which collects, organises and disseminates data on biological macromolecular structures.
- **DOI:** The International DOI Foundation (IDF), a not-for-profit membership organization that is the governance and management body for the federation of Registration Agencies providing Digital Object Identifier (DOI) services and registration, and is the registration authority for the ISO standard (ISO 26324) for the DOI system.
- **HAL:** HAL is an open archive where authors can deposit scholarly documents from all academic fields.
- **OLS:** The Ontology Lookup Service (OLS) is a repository for biomedical ontologies that aims to provide a single point of access to the latest ontology versions.

1.2.6 Tips

Use requests-cache

The `requests-cache` library is installed with `dabeplich`.

Using the Monkey-patching feature of the `requests-cache` library, caching will be used for all requests.

```
import requests_cache

requests_cache.install_cache(backend="memory")
```

Note: By default, it uses a `sqlite` db but you can also use `memory`, `redis` or `mongodb` instead.

1.2.7 Development environment

This page contains some guidelines to set up your environment to contribute to the project.

Install package for development

Packaging and organization of the library is done using `Poetry`

```
poetry install
```

This will install both dependencies needed for the library and for the development of the library.

Quality and tests

Quality and tests can be run using tasks. To obtain the full list:

```
poetry run inv --list
```

Quality

Code is formatted using `Black`. To run it:

```
poetry run inv quality.black-format
```

Documentation style is checked using `pydocstyle`. To run it:

```
poetry run inv quality.docstyle
```

Finally linting is done with `Flake8`. To run it:

```
poetry run inv quality.lint
```

Unit tests

To run all the unit tests you can use the following command from the root of the project:

```
poetry run inv tests.unit
```

Note: `-s` and `-v` options are used to get more details about the tests and redirect all output to stdout.

You can also only run selected tests by specifying the name of the test file:

```
poetry run pytest -s -v tests/dabeploch/parsers/
```

Note: You can find more information in the [pytest](#) documentation.

1.2.8 Contribution Flow

Every contributions need to go through a [Pull request](#) either from your own fork or a branch of the main repository.

Issues

To keep track and always have a dedicated space for discussions, we highly recommend to start your contribution through an issue on github.

Available issues

You can first have a look at the list of opened and available [issues](#).

Note: Issues can be filtered user a various number of labels. For instance if you want to add a new API connector, you might want to use the `new api` one.

Create an new issue

If you have a new idea that is not listed in the current [issues](#), Please feel free to create an new [issue](#).

1.2.9 Model description of the response

Warning: This step is optional and is required only if the related API does not return JSON format. Its main purpose at the moment is to help you build parsers.

Models description are done using [pydantic](#). In brief, it uses Python type annotations to perform validations. Please check to [Pydantic models documentation](#) for more details.

Note: This step is done manually at the moment, but we could imagine using or creating a tool that generates, in a semi-automatic manner the base structure for the response based on a openapi specification.

All the models are present in the `models` module of the dabeplich library.

Describing your response should be pretty straightforward and is only a matter of defining the structure of it. Let's have a look at a simple example.

Example

Let's say your API return some information about a gene:

```
{
  'gene_id': 'gene-3529',
  'name': 'Illestere',
  'sequence': 'acgtatcgaacagcatgcatgt'
}
```

You could therefore describe your response as followed:

```
from pydantic import BaseModel

class GeneModel(BaseModel):
    gene_id: str
    name: str
    sequence: str
```

It is as simple as that!

Note: Structure can be way more complex, with nested structure, regex validation, non mandatory fields... You can find all you need on the [Pydantic models documentation](#).

1.2.10 Parsers

Warning: This step is optional and is required only if the related API does not return JSON format.

Some API does not follow any standard and it can be complex to deal with its content. Thus, you might need to build some parsers to make the response stick to the model you described for the API

Parser structure

Parsers are build on the *model previously described* for your API. An abstract class is available to give you the main lines to build your parser:

Base with abstract classes for all parsers.

```
class dabeplich.parsers.base.BaseListParser(content_response)  
    Bases: abc.ABC
```

Base structure for parsers.

```
__init__(content_response)  
    Instantiate your parser on the response.
```

Parameters *content_response* – content response from the API

```
model  
    alias of pydantic.main.BaseModel
```

```
abstract parse()  
    Perform parsing of the content_response.
```

```
property validated_model  
    Retrieve entry validated with the model.
```

Returns Validated entry.

Return type BaseModel

```
class dabeplich.parsers.base.BaseParser(content_response)  
    Bases: abc.ABC
```

Base structure for parsers.

```
__init__(content_response)  
    Instantiate your parser on the response.
```

Parameters *content_response* – content response from the API

```
model  
    alias of pydantic.main.BaseModel
```

```
abstract parse()  
    Perform parsing of the content_response.
```

```
property validated_entry  
    Retrieve entry validated with the model.
```

Returns Validated entry.

Return type BaseModel

Note: We are aware that some API needs more specific and complex parsing, but the idea is to be perform parsing with the `parse` method and obtain the validated data with `validated_entry` property.

Test your parser

You can add your unit tests for your parser within `/tests/dabeplich/parsers` directory. You can have a look at the tests for kegg to help you build your own tests.

1.2.11 API connector

New API = New Module

When adding a new API connector, please create a new module within `dabeplich` with the api name. (e.g. `dabeplich/myawesomeapi.py`).

Note: To ease import while using `dabeplich`, do not forget to import your new classes to `dabeplich/__init__.py`.

Creating your API connector

Base Structure

All API connectors try to follow the same structure and therefore has to inherit from `dabeplich.base.BaseAPI`.

Here is the structure of the class:

```
class dabeplich.base.BaseAPI
    Bases: object

    Base class to build your API connector.

    BASE_URL = ''

    HEADERS = {'Accept':  '*/*', 'Content-type':  'application/json'}

    ROUTE = ''

    SESSION
        alias of requests.sessions.Session
```

When adding a new API connector, you inherit from the `BaseAPI` class and then overload `BASE_URL` with the base url of your API. From this new class you can create as many child classes as routes the API provides and overload `ROUTE` class attribute.

Then, you need to add methods to perform requests (GET, POST...) and the approach will depend on the API your are adding:

- *Adding an API returning JSON*
- *Adding an API based on a Parser*

Example

Let's say we want to add our new awesome API which has `https://awesomebioinfo.com/` as base url and contains two routes:

- `https://awesomebioinfo.com/genes`: obtain information about genes
- `https://awesomebioinfo.com/organisms`: obtain information about organisms

Here is the code that we write to add this API in `dabeplech/awesomebioinfo.py` file:

```
from dabeplech.base import BaseAPI

class AwesomebioinfoAPI(BaseAPI):
    BASE_URL = "https://awesomebioinfo.com/"

class AwesomebioinfoGenesAPI(AwesomebioinfoAPI):
    ROUTE = "genes/"

class AwesomebioinfoOrganismsAPI(AwesomebioinfoAPI):
    ROUTE = "organisms/"
```

Warning: As it is, your connectors won't work since no methods are defined to perform requests. This question is addressed below.

Once this module has been created, add it to the `dabeplech/__init__.py` file so that it can be imported. In this case, the code added at the bottom of this file would be:

```
from .awesomebioinfo import ( #noqa
    AwesomebioinfoAPI,
    AwesomebioinfoGenesAPI,
    AwesomebioinfoOrganismsAPI,
)
```

Adding an API returning JSON

A list of Mixins is available to build your API connectors. You just need to perform multiple inheritance for your new classes with the corresponding Mixins to add the methods. Here is a description of the different Mixins:

class `dabeplech.base.LISTMixin`

Bases: `object`

Corresponds to a GET that retrieve list of items.

list (*params=None*)

Perform GET request to the service.

Parameters *params* – query params for the request

Return type *list*

class `dabeplech.base.GETMixin`

Bases: `object`

Corresponds to a GET that retrieve one item from its ID.

get (*entry_id, params=None*)

Perform GET request to the service.

Parameters

- **entry_id** – ID of the entry you want to retrieve
- **params** – query params for the request

Return type dict**class** dabeplich.base.POSTMixin

Bases: object

Corresponds to a POST operation.

post (data)

Perform POST request to the service.

Args data: data to send in the body of your POST**Return type** dict**class** dabeplich.base.PUTMixin

Bases: object

Corresponds to a PUT operation.

put (data, entry_id=None)

Perform PUT request to the service.

Parameters

- **data** – data to send in the body of your PUT
- **entry_id** – ID of the entry you want to update

Return type Union[dict, list]**Example**

From the previous example, we consider the two endpoints allow to obtain list of genes and organisms, but also retrieve an item based on its ID. The code becomes:

```
from dabeplich.base import BaseAPI, LISTMixin, GETMixin

class AwesomebioinfoAPI(BaseAPI, LISTMixin, GETMixin):
    BASE_URL = "https://awesomebioinfo.com/"

class AwesomebioinfoGenesAPI(AwesomebioinfoAPI):
    ROUTE = "genes/"

class AwesomebioinfoOrganismsAPI(AwesomebioinfoAPI):
    ROUTE = "organisms/"
```

Now you can use your api connector (considering classes are added to dabeplich/__init__.py):

```
from dabeplich import AwesomebioinfoGenesAPI

api = AwesomebioinfoGenesAPI()

# Get all genes
all_genes = api.get_all()
```

(continues on next page)

(continued from previous page)

```
# Get one gene with its ID
the_gene = api.get('my-fav-gene')
```

Adding an API based on a Parser

There is not automatic way to help you use your parsers while adding an API connector using a parser to structure the response into JSON.

At the moment, please refer to the `dabeplich.kegg` module for examples using parsers.

Note: A more abstracted way of dealing with parser will be extracted in the future when needed. The Kegg case being particular, it needs its own methods.

Adding an API based on a Scraper

There is not automatic way to help you use your scrapers while adding an API connector using a scraper to structure the response into JSON.

At the moment, please refer to the `dabeplich.ncbi_taxonomy` module for examples using scrapers.

1.2.12 Completing the Documentation

In order to increase visibility while adding a new API connector to the library, we also encourage you add information to the documentation. For this you can edit the following pages:

- `user_guide/supported_api.rst`: add it to the list with a sentence to describe it.
- `user_guide/api_services.rst`: use `automodule` to add automatically generated documentation of the new API connector.

Note: Adding information on the present documentation also help increasing visibility of the added API.

1.2.13 Code of Conduct

Note: The following Code of Conduct was inspired from this [Gist](#).

Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, gender identity and expression, level of experience, nationality, personal appearance, race, religion, or sexual identity and orientation.

Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

1.2.14 API Services

KEGG

KEGGAPI()

KEGG API service (<http://rest.kegg.jp>).

dabepilech.kegg.KEGGAPI

```
class dabepilech.kegg.KEGGAPI
    KEGG API service (http://rest.kegg.jp).

    __init__()
        Instantiate your API connector.
```

Methods

<code><i>__init__</i>()</code>	Instantiate your API connector.
<code>find(database, query[, get_model])</code>	Perform KEGG FIND operation.
<code>get(entry_id[, get_model])</code>	Perform KEGG GET operation.
<code>link_db(target_db, source_db[, get_model])</code>	Perform KEGG LINK db operation.
<code>link_entries(target_db, db_entries[, get_model])</code>	Perform KEGG LINK entries operation.
<code>list(database[, get_model])</code>	Perform KEGG LIST operation.

Attributes

<code>ALLOWED_DATABASES</code>
<code>BASE_URL</code>
<code>HEADERS</code>
<code>ROUTE</code>

TogoWS

<code><i>TogoWSAPI</i>()</code>	TogoWS API service (http://togows.org).
<code><i>TogoWSEntryAPI</i>(database[, entry_format])</code>	TogoWS API service for entries.

dabepilech.togows.TogoWSAPI

```
class dabepilech.togows.TogoWSAPI
    TogoWS API service (http://togows.org).

    __init__()
        Instantiate your API connector.
```

Methods

<code>__init__()</code>	Instantiate your API connector.
<code>get(entry_id[, params])</code>	Perform GET request to the service.

Attributes

<code>BASE_URL</code>
<code>HEADERS</code>
<code>ROUTE</code>

dabeplich.togows.TogoWSEntryAPI

class dabeplich.togows.TogoWSEntryAPI (*database, entry_format='json'*)
TogoWS API service for entries.

__init__ (*database, entry_format='json'*)
Instantiate by choosing db and format for your response.

Parameters

- **database** – selected target database (list in self.DATABASES)
- **entry_format** – format for the response

Methods

<code>__init__(database[, entry_format])</code>	Instantiate by choosing db and format for your response.
<code>get(entry_id)</code>	Perform GET operation for an entry.
<code>get_field(entry_id, field)</code>	Perform GET field operation for an entry.

Attributes

<code>BASE_URL</code>
<code>DATABASES</code>
<code>FORMATS</code>
<code>HEADERS</code>
<code>ROUTE</code>

continues on next page

Table 8 – continued from previous page

TYPE

PDBe

<i>PDBeAPI()</i>	Base for PDBe API (https://www.ebi.ac.uk/pdbe/api/).
<i>PDBeUniprotMappingAPI()</i>	PDBe API for uniprot mapping route.
<i>PDBePFAMMappingAPI()</i>	PDBe API for pfam mapping route.

dabepilech.pdbe.PDBeAPI**class** dabepilech.pdbe.PDBeAPIBase for PDBe API (<https://www.ebi.ac.uk/pdbe/api/>).**__init__**()

Instantiate your API connector.

Methods

__init__ ()	Instantiate your API connector.
get (entry_id[, params])	Perform GET request to the service.

Attributes

BASE_URL
HEADERS
ROUTE

dabepilech.pdbe.PDBeUniprotMappingAPI**class** dabepilech.pdbe.PDBeUniprotMappingAPI

PDBe API for uniprot mapping route.

__init__()

Instantiate your API connector.

Methods

<code>__init__()</code>	Instantiate your API connector.
<code>get(entry_id[, params])</code>	Perform GET request to the service.

Attributes

<code>BASE_URL</code>
<code>HEADERS</code>
<code>ROUTE</code>

dabepilech.pdbe.PDBePFAMMappingAPI

class dabepilech.pdbe.PDBePFAMMappingAPI
 PDBe API for pfam mapping route.

`__init__()`
 Instantiate your API connector.

Methods

<code>__init__()</code>	Instantiate your API connector.
<code>get(entry_id[, params])</code>	Perform GET request to the service.

Attributes

<code>BASE_URL</code>
<code>HEADERS</code>
<code>ROUTE</code>

DOI

<code>DOIAPI()</code>	DOI API service (https://dx.doi.org).
-----------------------	--

dabeplech.doi.DOIAPI

```
class dabeplech.doi.DOIAPI
    DOI API service (https://dx.doi.org).

    __init__()
        Instantiate your API connector.
```

Methods

<code>__init__()</code>	Instantiate your API connector.
<code>get(entry_id[, params])</code>	Perform GET request to the service.

Attributes

<code>BASE_URL</code>
<code>HEADERS</code>
<code>ROUTE</code>

HAL

<code>HALAPI()</code>	HAL API service (https://api.archives-ouvertes.fr/search/).
---------------------------------------	--

dabeplech.hal.HALAPI

```
class dabeplech.hal.HALAPI
    HAL API service (https://api.archives-ouvertes.fr/search/).

    __init__()
        Instantiate your API connector.
```

Methods

<code>__init__()</code>	Instantiate your API connector.
<code>find(doi)</code>	Perform FIND operation.
<code>get(hal_id)</code>	Perform GET operation.

Attributes

BASE_URL

HEADERS

ROUTE

OLS

<i>OLSGOAPI()</i>	OLS GO API service (https://www.ebi.ac.uk/ols/api/ontologies/go/terms/).
<i>OLSNCBITaxonomyAPI()</i>	OLS NCBI Taxonomy API service (https://www.ebi.ac.uk/ols/api/ontologies/ncbitaxon/terms/).

dabeplech.ols.OLSGOAPI

class dabeplech.ols.OLSGOAPI

OLS GO API service (<https://www.ebi.ac.uk/ols/api/ontologies/go/terms/>).

__init__()

Instantiate your API connector.

Methods

<i>__init__</i> ()	Instantiate your API connector.
<i>get</i> (entry_id[, params])	Perform GET request to the service.
<i>get_term</i> (entry_id)	Perform GET request to the service using the ontology term ID only.

Attributes

BASE_URL

HEADERS

ROUTE

dabeplich.ols.OLSNCBITaxonomyAPI**class** dabeplich.ols.OLSNCBITaxonomyAPIOLS NCBI Taxonomy API service (<https://www.ebi.ac.uk/ols/api/ontologies/ncbitaxon/terms/>).**__init__**()

Instantiate your API connector.

Methods

<code>__init__()</code>	Instantiate your API connector.
<code>get(entry_id[, params])</code>	Perform GET request to the service.

Attributes

<code>BASE_URL</code>
<code>HEADERS</code>
<code>ROUTE</code>

Base

Base items to help you build your API connectors:

<code>BaseAPI()</code>	Base class to build your API connector.
<code>LISTMixin()</code>	Corresponds to a GET that retrieve list of items.
<code>GETMixin()</code>	Corresponds to a GET that retrieve one item from its ID.
<code>POSTMixin()</code>	Corresponds to a POST operation.
<code>PUTMixin()</code>	Corresponds to a PUT operation.

dabeplich.base.BaseAPI**class** dabeplich.base.BaseAPI

Base class to build your API connector.

__init__()

Instantiate your API connector.

Methods

<code>__init__()</code>	Instantiate your API connector.
-------------------------	---------------------------------

Attributes

<code>BASE_URL</code>
<code>HEADERS</code>
<code>ROUTE</code>

dabeplech.base.LISTMixin

class dabeplech.base.LISTMixin

Corresponds to a GET that retrieve list of items.

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

Methods

<code>__init__()</code>	Initialize self.
<code>list([params])</code>	Perform GET request to the service.

dabeplech.base.GETMixin

class dabeplech.base.GETMixin

Corresponds to a GET that retrieve one item from its ID.

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

Methods

<code>__init__()</code>	Initialize self.
<code>get(entry_id[, params])</code>	Perform GET request to the service.

dabeplich.base.POSTMixin**class** dabeplich.base.POSTMixin

Corresponds to a POST operation.

__init__()

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

Methods

<code>__init__()</code>	Initialize self.
<code>post(data)</code>	Perform POST request to the service.

dabeplich.base.PUTMixin**class** dabeplich.base.PUTMixin

Corresponds to a PUT operation.

__init__()

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

Methods

<code>__init__()</code>	Initialize self.
<code>put(data[, entry_id])</code>	Perform PUT request to the service.

1.2.15 Parsers

Different available parsers.

Base

Abstract classes to give a base for parsers.

<code>BaseParser(content_response)</code>	Base structure for parsers.
<code>BaseListParser(content_response)</code>	Base structure for parsers.

dabeplich.parsers.base.BaseParser**class** dabeplich.parsers.base.BaseParser(*content_response*)

Base structure for parsers.

__init__(*content_response*)

Instantiate your parser on the response.

Parameters **content_response** – content response from the API

Methods

<code>__init__(content_response)</code>	Instantiate your parser on the response.
<code>parse()</code>	Perform parsing of the <code>content_response</code> .

Attributes

<code>validated_entry</code>	Retrieve entry validated with the model.
------------------------------	--

dabeplich.parsers.base.BaseListParser

class dabeplich.parsers.base.**BaseListParser** (*content_response*)

Base structure for parsers.

`__init__` (*content_response*)

Instantiate your parser on the response.

Parameters `content_response` – content response from the API

Methods

<code>__init__(content_response)</code>	Instantiate your parser on the response.
<code>parse()</code>	Perform parsing of the <code>content_response</code> .

Attributes

<code>validated_model</code>	Retrieve entry validated with the model.
------------------------------	--

KEGG

<code>KeggLinkParser(*args, **kwargs)</code>	Parser for LINK operation from KEGG API.
<code>KeggOrthologyListParser(content_response)</code>	Parser for list of KEGG ko from KEGG API (http://rest.kegg.jp/list/ko).
<code>KeggPathwayListParser(content_response)</code>	Parser for list of KEGG pathways (map) from KEGG API (http://rest.kegg.jp/list/pathway).
<code>KeggModuleListParser(content_response)</code>	Parser for list of KEGG modules (M) from KEGG API (http://rest.kegg.jp/list/module).
<code>KeggModuleParser(content_response)</code>	Parser for KEGG module <i>plain text</i> result from KEGG API.
<code>KeggOrthologyParser(content_response)</code>	Parser for KEGG KO <i>plain text</i> result from KEGG API.
<code>KeggPathwayParser(content_response)</code>	Parser for KEGG pathway <i>plain text</i> result from KEGG API.

dabepilech.parsers.kegg.KeggLinkParser**class** dabepilech.parsers.kegg.**KeggLinkParser** (*args, **kwargs)

Parser for LINK operation from KEGG API.

__init__ (*args, **kwargs)

Instantiate your parser on the KEGG response.

Parameters **content_response** – content response from the API**Methods**

__init__ (*args, **kwargs)	Instantiate your parser on the KEGG response.
parse ()	Perform parsing of the content_response .

Attributes

validated_entry	Retrieve entry validated with the model.
------------------------	--

dabepilech.parsers.kegg.KeggOrthologyListParser**class** dabepilech.parsers.kegg.**KeggOrthologyListParser** (content_response)Parser for list of KEGG ko from KEGG API (<http://rest.kegg.jp/list/ko>).**__init__** (content_response)

Instantiate your parser on the response.

Parameters **content_response** – content response from the API**Methods**

__init__ (content_response)	Instantiate your parser on the response.
parse ()	Perform parsing of the content_response .

Attributes

validated_model	Retrieve entry validated with the model.
------------------------	--

dabeplich.parsers.kegg.KeggPathwayListParser

class dabeplich.parsers.kegg.**KeggPathwayListParser** (*content_response*)
 Parser for list of KEGG pathways (map) from KEGG API (<http://rest.kegg.jp/list/pathway>).

__init__ (*content_response*)
 Instantiate your parser on the response.

Parameters **content_response** – content response from the API

Methods

<u>__init__</u> (<i>content_response</i>)	Instantiate your parser on the response.
parse ()	Perform parsing of the <i>content_response</i> .

Attributes

validated_model	Retrieve entry validated with the model.
------------------------	--

dabeplich.parsers.kegg.KeggModuleListParser

class dabeplich.parsers.kegg.**KeggModuleListParser** (*content_response*)
 Parser for list of KEGG modules (M) from KEGG API (<http://rest.kegg.jp/list/module>).

__init__ (*content_response*)
 Instantiate your parser on the response.

Parameters **content_response** – content response from the API

Methods

<u>__init__</u> (<i>content_response</i>)	Instantiate your parser on the response.
parse ()	Perform parsing of the <i>content_response</i> .

Attributes

validated_model	Retrieve entry validated with the model.
------------------------	--

dabepilech.parsers.kegg.KeggModuleParser

class dabepilech.parsers.kegg.**KeggModuleParser** (*content_response*)
Parser for KEGG module *plain text* result from KEGG API.

__init__ (*content_response*)

Instantiate your parser on the KEGG response.

Parameters **content_response** – content response from the API

Methods

<code>__init__(content_response)</code>	Instantiate your parser on the KEGG response.
<code>parse()</code>	Perform parsing of the text content into the model defined for KEGG orthology.

Attributes

<code>validated_entry</code>	Retrieve KEGG entry validated with the model.
------------------------------	---

dabepilech.parsers.kegg.KeggOrthologyParser

class dabepilech.parsers.kegg.**KeggOrthologyParser** (*content_response*)
Parser for KEGG KO *plain text* result from KEGG API.

__init__ (*content_response*)

Instantiate your parser on the KEGG response.

Parameters **content_response** – content response from the API

Methods

<code>__init__(content_response)</code>	Instantiate your parser on the KEGG response.
<code>parse()</code>	Perform parsing of the text content into the model defined for KEGG orthology.

Attributes

<code>validated_entry</code>	Retrieve KEGG entry validated with the model.
------------------------------	---

dabeplich.parsers.kegg.KeggPathwayParser

class dabeplich.parsers.kegg.**KeggPathwayParser** (*content_response*)

Parser for KEGG pathway *plain text* result from KEGG API.

___**init**___ (*content_response*)

Instantiate your parser on the KEGG response.

Parameters **content_response** – content response from the API

Methods

___ <i>init</i> ___(content_response)	Instantiate your parser on the KEGG response.
parse()	Perform parsing of the text content into the model defined for KEGG orthology.

Attributes

validated_entry	Retrieve KEGG entry validated with the model.
-----------------	---

1.2.16 Changelogs

Summary of developments of dabeplich library.

v0.5**v0.5.0**

- Update docs and docstring format (pydocstyle and black)
- Reformat and changes import path models and scrappers

v0.4**v0.4.0**

- Add an endpoint to query Ontology Lookup Service <https://www.ebi.ac.uk/ols/index> (<https://github.com/motleystate/dabeplich/pull/30>)
 - a first `get_term` method to query by ontology term ID

v0.3

v0.3.0

- Add an endpoint to query the HAL (<https://hal.archives-ouvertes.org>) endpoint to retrieve information in the JSON format. (<https://github.com/motleystate/dabeplich/pull/29>)

v0.2

v0.2.0

- Add an endpoint to query the DOI (<https://dx.doi.org>) endpoint to retrieve information in the JSON format. (<https://github.com/motleystate/dabeplich/pull/28>)

v0.1

v0.1.1

- Fix bug and deal with variant of NCBI taxonomy item page (e.g. `tax_id=12345`)

v0.1.0

- Add scrapper for NCBI taxonomy (<https://github.com/motleystate/dabeplich/pull/24>)
- Add `NCBITaxonomyScrapAPI` that mimics API behaviour to retrieve hierarchy information for a given `tax_id` (<https://github.com/motleystate/dabeplich/pull/24>)
- add first PDBe REST endpoints (<https://github.com/motleystate/dabeplich/pull/21>)

v0.0

v0.0.6

- LIST of entries described in KEGG models only contains the necessary fields (<https://github.com/motleystate/dabeplich/pull/20>)
- deal with empty responses (200) from *FIND* operation on KEGG API (<https://github.com/motleystate/dabeplich/pull/19>)

v0.0.5

- `FIND` works and gives a json output for pathway and ko. It can be used using `.find(database, query)` (<https://github.com/motleystate/dabeplich/pull/16>)
- `LINK` has been splitted in two methods (there are using the same endpoint, but having two different methods make more sense). (<https://github.com/motleystate/dabeplich/pull/16>)
 - `link_db(target_db, source_db)`: allows retrieval of database to database cross-references
 - `link_entries(target_db, dbentries)`: allows retrieval for a selected number of entries
- replace `get_all()` by `list()` for LIST operation on KEGG API

- Add model description and API for KEGG module (<https://github.com/motleystate/dabeplech/pull/17>)
- change name to `dabeplech`

v0.0.4

- Replace models for pathway and ko list by simpler model

v0.0.3

- Add parser and api for KEGG pathway list (<https://github.com/khillion/dabeplech/pull/10>)
- Add parser and api for KEGG ko list (<https://github.com/khillion/dabeplech/pull/10>)
- Change `name` attribute to `names` for KEGG
- Handle reference with no pubmed ID (<https://github.com/khillion/dabeplech/pull/7>)

v0.0.2

- Add parser for KEGG pathway response

v0.0.1

This is the first release of `dabeplech`:

- Parser for KEGG orthology response
- Base structure for KEGG API
 - Return structured dict if parser available
 - If no parser available, return raw response from KEGG API
- Base structure for TogoWS API

PYTHON MODULE INDEX

d

`dabeplech.parsers.base`, [10](#)

Symbols

`__init__()` (*dabep`le`ch.base.BaseAPI* method), 22
`__init__()` (*dabep`le`ch.base.GETMixin* method), 23
`__init__()` (*dabep`le`ch.base.LISTMixin* method), 23
`__init__()` (*dabep`le`ch.base.POSTMixin* method), 24
`__init__()` (*dabep`le`ch.base.PUTMixin* method), 24
`__init__()` (*dabep`le`ch.doi.DOIAPI* method), 20
`__init__()` (*dabep`le`ch.hal.HALAPI* method), 20
`__init__()` (*dabep`le`ch.kegg.KEGGAPI* method), 16
`__init__()` (*dabep`le`ch.ols.OLSGOAPI* method), 21
`__init__()` (*dabep`le`ch.ols.OLSNCBITaxonomyAPI* method), 22
`__init__()` (*dabep`le`ch.parsers.base.BaseListParser* method), 10, 25
`__init__()` (*dabep`le`ch.parsers.base.BaseParser* method), 10, 24
`__init__()` (*dabep`le`ch.parsers.kegg.KeggLinkParser* method), 26
`__init__()` (*dabep`le`ch.parsers.kegg.KeggModuleListParser* method), 27
`__init__()` (*dabep`le`ch.parsers.kegg.KeggModuleParser* method), 28
`__init__()` (*dabep`le`ch.parsers.kegg.KeggOrthologyListParser* method), 26
`__init__()` (*dabep`le`ch.parsers.kegg.KeggOrthologyParser* method), 28
`__init__()` (*dabep`le`ch.parsers.kegg.KeggPathwayListParser* method), 27
`__init__()` (*dabep`le`ch.parsers.kegg.KeggPathwayParser* method), 29
`__init__()` (*dabep`le`ch.pdbe.PDBeAPI* method), 18
`__init__()` (*dabep`le`ch.pdbe.PDBePFAMMappingAPI* method), 19
`__init__()` (*dabep`le`ch.pdbe.PDBeUniprotMappingAPI* method), 18
`__init__()` (*dabep`le`ch.togows.TogoWSAPI* method), 16
`__init__()` (*dabep`le`ch.togows.TogoWSEntryAPI* method), 17

B

`BASE_URL` (*dabep`le`ch.base.BaseAPI* attribute), 11

`BaseAPI` (*class in dabep`le`ch.base*), 11, 22
`BaseListParser` (*class in dabep`le`ch.parsers.base*), 10, 25
`BaseParser` (*class in dabep`le`ch.parsers.base*), 10, 24

D

`dabeplech.parsers.base`
 module, 10
`DOIAPI` (*class in dabep`le`ch.doi*), 20

G

`get()` (*dabep`le`ch.base.GETMixin* method), 12
`GETMixin` (*class in dabep`le`ch.base*), 12, 23

H

`HALAPI` (*class in dabep`le`ch.hal*), 20
`HEADERS` (*dabep`le`ch.base.BaseAPI* attribute), 11

K

`KEGGAPI` (*class in dabep`le`ch.kegg*), 16
`KeggLinkParser` (*class in dabep`le`ch.parsers.kegg*), 26
`KeggModuleListParser` (*class in dabep`le`ch.parsers.kegg*), 27
`KeggModuleParser` (*class in dabep`le`ch.parsers.kegg*), 28
`KeggOrthologyListParser` (*class in dabep`le`ch.parsers.kegg*), 26
`KeggOrthologyParser` (*class in dabep`le`ch.parsers.kegg*), 28
`KeggPathwayListParser` (*class in dabep`le`ch.parsers.kegg*), 27
`KeggPathwayParser` (*class in dabep`le`ch.parsers.kegg*), 29

L

`list()` (*dabep`le`ch.base.LISTMixin* method), 12
`LISTMixin` (*class in dabep`le`ch.base*), 12, 23

M

`model` (*dabep`le`ch.parsers.base.BaseListParser* attribute), 10

`model` (*dabeplech.parsers.base.BaseParser* attribute),
10
`module`
`dabeplech.parsers.base`, 10

O

`OLSGOAPI` (*class in dabeplech.ols*), 21
`OLSNCBITaxonomyAPI` (*class in dabeplech.ols*), 22

P

`parse()` (*dabeplech.parsers.base.BaseListParser*
method), 10
`parse()` (*dabeplech.parsers.base.BaseParser* method),
10
`PDBeAPI` (*class in dabeplech.pdbe*), 18
`PDBePFAMMappingAPI` (*class in dabeplech.pdbe*), 19
`PDBeUniprotMappingAPI` (*class in*
dabeplech.pdbe), 18
`post()` (*dabeplech.base.POSTMixin* method), 13
`POSTMixin` (*class in dabeplech.base*), 13, 24
`put()` (*dabeplech.base.PUTMixin* method), 13
`PUTMixin` (*class in dabeplech.base*), 13, 24

R

`ROUTE` (*dabeplech.base.BaseAPI* attribute), 11

S

`SESSION` (*dabeplech.base.BaseAPI* attribute), 11

T

`TogoWSAPI` (*class in dabeplech.togows*), 16
`TogoWSEntryAPI` (*class in dabeplech.togows*), 17

V

`validated_entry()`
(*dabeplech.parsers.base.BaseParser* prop-
erty), 10
`validated_model()`
(*dabeplech.parsers.base.BaseListParser*
property), 10