craedl-sdk-python

Release 1.0.3

Mar 24, 2022

Contents

CHAPTER 1

Introduction

The Craedl Python SDK (Software Development Kit) enables Craedl.org users to access their Craedl accounts using the Python programming language. This provides a mechanism for using Craedl on computers without access to a web browser (such as a high-performance computing cluster) and to automate common Craedl project manipulations (such as file uploads and downloads) within a Python script.

CHAPTER 2

Quick start

Get started with the Craedl Python SDK by obtaining it via PyPI:

```
pip install craedl
```

Log into your Craedl account at Craedl.org and generate an API access token by clicking the key in your profile view. Copy your token and paste it when prompted after running one of the following commands:

(A) Configure your account through a system shell

```
python -m craedl
```

(B) Configure your account through an interactive Python interpreter

```
import craedl
craedl.configure()
```

Now you can use Python to access your Craedl, for example:

```
import craedl
profile = craedl.auth()
for craedl in profile.craedls:
    print(craedl)
```

CHAPTER 3

Documentation

3.1 The basics

The Craedl Python SDK is, ultimately, a wrapper around the Craedl RESTful API. Through the RESTful API, you can do just about anything in code that you can do in the Craedl web site, as long as you don't mind composing commands like this one and then manually parsing the JSON response:

```
curl -H "Authorization: Bearer RqjxUjHwuoov0LvVQV1bEuGLOftwEfOiLHXaGxzh" https://api.
↪craedl.org/profile/whoami/
```

The Craedl Python SDK takes care of all of this messy HTTP request formatting and JSON parsing for you, replacing it with an intuitive Python module that you can use to easily script your Craedl usage.

The ability to script your Craedl usage can be beneficial if you commonly perform repetitive tasks like uploading the results of an experiment. It can be mission-critical if you want to access Craedl through a computing resource that doesn't have a GUI (because then it's impossible to access Craedl through a web browser).

Follow these steps to get started using the Craedl Python SDK.

3.1.1 Obtain the Craedl Python SDK module

Obtain the Craedl Python SDK from PyPI:

```
pip install craedl
```

You only have to perform this step once (on a particular computer and/or virtual environment).

3.1.2 Configure your authentication token

Retrieve your API access token from your Craedl account by logging into Craedl.org and clicking the key tab in your profile. Generate a token and pass it to one of the following commands when prompted:

(A) Configure your account through a system shell

```
python -m craedl
```

(B) Configure your account through an interactive Python interpreter

```
import craedl
craedl.configure()
```

This token will remain active indefinitely. Should you have reason to worry that your token has been compromised, use the interface in your Craedl profile to revoke the compromised token, generate a new one, and re-run the command above to enable your Craedl Python SDK authentication.

3.1.3 Use the Craedl Python SDK

Now you're ready to write a Craedl Python script. To begin, you must always import the Craedl Python SDK and get your profile:

```
import craedl
profile = craedl.auth()
```

From here, you can put together the building blocks documented in *Craedl Python SDK documentation*. If you're just getting started, you may find our *Examples* helpful.

3.2 Examples

3.2.1 See Craedls

```
import craedl
profile = craedl.auth()

# print your craedl tree
for craedl in profile.craedls:
  print(craedl)

# get a particular craedl
craedl = profile.get_craedl('craedl-slug', 1)
```

Printing the Craedl tree generates output such as:

```
Test Craedl [craedl-slug:1]
  - Child 1 [craedl-slug:2]
      - Child 1a [craedl-slug:5]
          - Child 1a1 [craedl-slug:7]
      - Child 1b [craedl-slug:8]
      - Child 1c [craedl-slug:9]
  - Child 2 [craedl-slug:3]
      - Child 2a [craedl-slug:4]
```

where 'craedl-slug' is the Craedl's slug (visible in the URL through the web browser) and the number is the Craedl's ID. Getting a particular Craedl requires this slug and ID.

3.2.2 Download data

```
import craedl
profile = craedl.auth()

# get a craedl
craedl = profile.get_craedl('craedl-slug', 1)

# access the data in your craedl
root = craedl.get_data()

# get the contents of a particular directory
directory = root.get('path/to/data/in/Craedl')
children = directory.list()
for directories in children['dirs']:
  print(directories)
for files in children['files']:
  print(files)

# download the 0-th file's data in this directory
file_downloaded = files[0].download('path/on/local/computer/to/save/data')
```

3.2.3 Upload data

```
import craedl
profile = craedl.auth()

# get a craedl
craedl = profile.get_craedl('craedl-slug', 1)

# access the data in your craedl
root = craedl.get_data()

# get a particular directory
directory = root.get('path/to/data/in/Craedl')

# create a new directory inside directory
directory_new = directory.create_directory('new-directory-name')

# upload a new file into directory_new
directory_new = directory_new.upload(
    '/path/on/local/computer/to/read/data'
)
```

3.2.4 Upload directory recursively

```
import craedl
profile = craedl.auth()

# get a craedl
craedl = profile.get_craedl('craedl-slug', 1)

# access the data in your craedl
```

(continues on next page)

(continued from previous page)

```
root = craedl.get_data()

# get a particular directory
directory = root.get('path/to/data/in/Craedl')

# upload the directory recursively
# this incantation of upload() will pick up from where it left off
# if it is stopped for any reason
directory = directory.upload(
    '/path/on/local/computer/to/read/data',
    rescan=False, # ignores new children in directories already transferred
    output=True # outputs progress to STDOUT
)
```

3.3 Craedl Python SDK documentation

3.3.1 Core

Auth

class craedl.core.**Auth**(*host=None*)

This base class handles low-level RESTful API communications. Any class that needs to perform RESTful API communications should extend this class.

DELETE(*path*)

Handle a DELETE request.

Parameters **path** (*string*) – the RESTful API method path

Returns a dict containing the contents of the parsed JSON response or an HTML error string if the response does not have status 200

GET(*path*, *data=False*)

Handle a GET request.

Parameters

- **path** (*string*) – the RESTful API method path
- **data** (*boolean*) – whether the response is a data stream (default False)

Returns a dict containing the contents of the parsed JSON response, data stream, or an HTML error string if the response does not have status 200

POST(*path*, *data*, *filepath=None*)

Handle a POST request.

Parameters

- **path** (*string*) – the RESTful API method path
- **data** (*dict*) – the data to POST to the RESTful API method as described at https://api.craedl.org
- **filepath** (*string*) – the path to the file to be passed

Returns a dict containing the contents of the parsed JSON response or an HTML error string if the response does not have status 200

process_response(*response*)
Process the response from a RESTful API request.

Parameters **response** (*a response object*) – the RESTful API response

Returns a dict containing the contents of the parsed JSON response or an HTML error string if the response does not have status 200

Craedl

class craedl.core.**Craedl**(*host*, *slug*, *id=None*, *data={}*)
A Craedl. Get a Craedl from the API by passing *id*. Get a Craedl from an existing dictionary (for example, a child carried by a parent)without any network traffic by passing *data*.

get_data()
Retrieve the root directory associated with this Craedl.

Returns the root Inode of this Craedl

get_wiki(*revision=0*)
Retrieve the wiki for this Craedl.

Parameters **revision** (*int*) – the historical revision to view, moving backward through time (defaults to current revision)

Inode

class craedl.core.**Inode**(*craedl*, *id=None*, *data={}*)
A Craedl inode (directory or file) object. Get an Inode from the API by passing *id*. Get an Inode from an eisting dictionary (for example, a child carried by a parent) without any network traffic by passing *data*.

abspath()
Get a string representation of the absolute path for an inode.

Returns a string containing the absolute path for an inode

create_directory(*name*)
Create a new directory within the current directory.

Parameters **name** (*string*) – the name of the new directory

Returns the new Inode

delete()
Delete the current directory.

download(*save_path*, *rescan=True*, *output=False*, *accumulated_size=0*)
Download the contents of the current Inode into *save_path*. If the current Inode is a directory, download recursively; this generates a cache database file in the *save_path* that is used to enhance performance of retries and synchronizations.

Parameters

- **save_path** (*string*) – the path to the directory on your computer that will contain this file's data
- **rescan** (*boolean*) – whether to rescan the directories (defaults to True); ignores new children in already transferred directories if False
- **output** (*boolean*) – whether to print to STDOUT (defaults to False)

- **accumulated_size** – the total size of the download so far; primarily supports recursive download output messages

Returns a tuple containing the updated instance of this directory and the size of the download

download_recurse(*cache*, *save_path*, *rescan*, *output*, *accumulated_size*)
The recursive function that does the downloading. There is little reason to call this directly; use *Inode.download()* to start a recursive directory download.

Parameters

- **cache** (*Cache*) – the cache database
- **save_path** (*string*) – the path to the directory on your computer that will contain this file's data
- **rescan** (*boolean*) – whether to rescan the directories (defaults to True); ignores new children in already transferred directories if False
- **output** (*boolean*) – whether to print to STDOUT (defaults to False)
- **accumulated_size** – the amount of data that has been downloaded so far

Type integer

Returns a tuple containing the updated instance of this directory and the amount of data that has been downloaded by this recursion level and its children

get(*path*)
Get a particular directory or file. This may be an absolute or relative path.

Parameters **path** (*string*) – the directory or file path

Returns the requested directory or file

list()
List the contents of this directory.

Returns a dictionary containing a list of directories ('dirs') and a list of files ('files')

upload(*path*, *rescan=True*, *output=False*, *follow_symlinks=False*)
Upload to the current directory. If uploading a directory, this generates a cache database in the *path* that is used to enhanceperformance of retries and synchronizations.

Parameters

- **path** (*string*) – the local path to the file/directory to be uploaded
- **rescan** (*boolean*) – whether to rescan the directories (defaults to True); ignores new children in already transferred directories if False
- **output** (*boolean*) – whether to print to STDOUT (defaults to False)
- **follow_symlinks** (*boolean*) – whether to follow symlinks (default False)

Returns the uploaded Inode

upload_directory(*directory_path*, *rescan=True*, *follow_symlinks=False*, *output=False*)
Upload a new directory contained within this directory. It generates a cache database in the *directory_path* that is used to enhance performance of retries and synchronizations.

Parameters

- **directory_path** (*string*) – the path to the directory to be uploaded on your computer
- **follow_symlinks** (*boolean*) – whether to follow symlinks (default False)

- **rescan** (*boolean*) – whether to rescan the directories (defaults to True); ignores new children in already transferred directories if False
- **output** (*boolean*) – whether to print to STDOUT (defaults to False)

Returns the updated instance of the current directory

upload_directory_recurse(*cache*, *directory_path*, *rescan*, *follow_symlinks*, *output*, *accumulated_size*)

The recursive function that does the uploading. There is little reason to call this directly; use `Directory.upload_directory()` to start a directory upload.

Parameters

- **cache** (`Cache`) – the cache database
- **directory_path** (*string*) – the path to the directory on your computer that will contain this file's data
- **rescan** (*boolean*) – whether to rescan the directories (defaults to True); ignores new children in already transferred directories if False
- **follow_symlinks** (*boolean*) – whether to follow symlinks
- **output** (*boolean*) – whether to print to STDOUT
- **accumulated_size** – the amount of data that has been uploaded so far

Type int

Returns a tuple containing the updated instance of this directory and the amount of data that has been downloaded by this recursion level and its children

upload_file(*file_path*, *output=False*, *accumulated_size=0*)

Upload a new file contained within this directory.

Parameters

- **file_path** (*string*) – the path to the file to be uploaded on your computer
- **output** (*boolean*) – whether to print to STDOUT (defaults to False)
- **accumulated_size** (*int*) – the size that has accumulated prior to this upload (defaults to 0); this is entirely for output purposes

Returns a tuple with the updated instance of the current directory and the uploaded size

Profile

class craedl.core.**Profile**(*id=None*, *data=None*)

A Craedl profile object. Get a Profile from the API by passing *id*. Get a Profile from an existing dictionary without network traffic by passing *data*. Get your profile by passing no arguments.

get_craedl(*slug*, *id=None*)

Get a particular Craedl.

Parameters

- **slug** (*string*) – the Craedl slug
- **id** (*int*) – the Craedl id

Returns A Craedl instance

get_craedls()
Get a list of craedls to which this profile belongs.

Returns a list of craedls

3.3.2 Cache

Cache

class craedl.cache.**Cache**
This class handles all cache database operations. It is used largely in support of recursive file uploads and downloads.

It operates by storing a SHA1 hash representing the time stamps for children within a given directory and a status bit that indicates whether the contents were successfully transferred at the time the hash was created.

Index