clarifai Documentation

Release 2.4.2

Clarifai

Nov 20, 2018

First steps

CHAPTER 1

First steps

1.1 Installation Guide

Note: Generally, the python client works with Python 2.x and Python 3.x. But it is only tested against 2.7, 3.5, 3.6 and 3.7. Feel free to report any bugs you encounter in any other version.

1.1.1 Install the package

You can install the latest stable clarifai using pip (which is the canonical way to install Python packages). To install using `pip`, run:

```
pip install clarifai --upgrade
```

You can also install directly from Github (though we recommend the previous approach):

```
pip install git+git://github.com/Clarifai/clarifai-python.git
```

1.1.2 Configuration

Generate your Clarifai API key [on the API keys page](https://clarifai.com/developer/account/keys). The client uses it for authentication.

You can use three methods to pass the API key to your client (if you use more than one, the first in the following precedence order will be used):

1. Pass it to the `ClarifaiApp` constructor through the `api_key` parameter.
2. Set it as the `CLARIFAI_API_KEY` environment variable.
3. Place it in the `.clarifai/config` file using the following command (see exceptions for Windows below):

```
$ ./scripts/clarifai config
CLARIFAI_API_KEY: []: **********************************YQEd
```

Windows Users

For Windows users, running `./scripts/clarifai config` may fail when you try to configure the runtime environment. This is because Windows uses the file extension to determine executables and by default, file `clarifai` without file extension is non-executable. In order to run the command, you may want to launch it with the python interpreter.

```
C:\Python27\python.exe Scripts\clarifai config
CLARIFAI_API_KEY: []: **********************************YQEd
```

1.1.3 AWS Lambda Users

For AWS Lambda users, in order to use the library correctly, you are recommended to set an environmental variable *CLARIFAI_API_KEY* in the lambda function configuration, or "hardcode" the `api_key` parameter to the `ClarifaiApp` constructor.

1.2 Tutorial

Each of the examples below is a small independent code snippet within 10 lines that could work by copy and paste to a python source code file. By playing with them, you should be getting started with Clarifai API. For more information about the API, check the API Reference.

1.2.1 Predict Tutorial

Predict with Public Models

For more information on any of the public models, visit https://developer.clarifai.com/models

```
from clarifai.rest import ClarifaiApp

app = ClarifaiApp()

#General model
model = app.models.get('general-v1.3')

response = model.predict_by_url(url='https://samples.clarifai.com/metro-north.jpg')


#Travel model
model = app.models.get('travel-v1.0')

response = model.predict_by_url(url='https://samples.clarifai.com/travel.jpg')


#Food model
model = app.models.get('food-items-v1.0')
```

(continues on next page)

(continued from previous page)

```
response = model.predict_by_url(url='https://samples.clarifai.com/food.jpg')


#NSFW model
model = app.models.get('nsfw-v1.0')

response = model.predict_by_url(url='https://samples.clarifai.com/nsfw.jpg')


#Apparel model
model = app.models.get('apparel')

response = model.predict_by_url(url='https://samples.clarifai.com/apparel.jpg')


#Celebrity model
model = app.models.get('celeb-v1.3')

response = model.predict_by_url(url='https://samples.clarifai.com/celebrity.jpg')


#Demographics model
model = app.models.get('demographics')

response = model.predict_by_url(url='https://samples.clarifai.com/demographics.jpg')


#Face Detection model
model = app.models.get('face-v1.3')

response = model.predict_by_url(url='https://developer.clarifai.com/static/images/
↪model-samples/face-001.jpg')


#Focus Detection model
model = app.models.get('focus')

response = model.predict_by_url(url='https://samples.clarifai.com/focus.jpg')


#General Embedding model
model = app.models.get('general-v1.3', model_type='embed')

response = model.predict_by_url(url='https://samples.clarifai.com/metro-north.jpg')


#Logo model
model = app.models.get('logo')

response = model.predict_by_url(url='https://samples.clarifai.com/logo.jpg')


#Color model
model = app.models.get('color', model_type='color')

response = model.predict_by_url(url='https://samples.clarifai.com/wedding.jpg')
```

1.2.2 Feedback Tutorial

Concept model prediction

```
from clarifai.rest import ClarifaiApp
from clarifai.rest import FeedbackInfo

app = ClarifaiApp()

# positive feedback: this is a dog
m = app.models.get('general-v1.3')

m.send_concept_feedback(input_id='id1', url='https://samples.clarifai.com/dog2.jpeg',
                        concepts=['dog', 'animal'],
                        feedback_info=FeedbackInfo(output_id='OID',
                                                   session_id='SID',
                                                   end_user_id='UID',
                                                   event_type='annotation'))

# negative feedback: this is not a cat
m = app.models.get('general-v1.3')

m.send_concept_feedback(input_id='id1', url='https://samples.clarifai.com/dog2.jpeg',
                        not_concepts=['cat', 'kitty'],
                        feedback_info=FeedbackInfo(output_id='OID',
                                                   session_id='SID',
                                                   end_user_id='UID',
                                                   event_type='annotation'))

# all together: this is a dog but not a cat
m = app.models.get('general-v1.3')

m.send_concept_feedback(input_id='id1', url='https://samples.clarifai.com/dog2.jpeg',
                        concepts=['dog'], not_concepts=['cat', 'kitty'],
                        feedback_info=FeedbackInfo(output_id='OID',
                                                   session_id='SID',
                                                   end_user_id='UID',
                                                   event_type='annotation'))
```

Detection model prediction

```
from clarifai.rest import ClarifaiApp
from clarifai.rest import FeedbackInfo
from clarifai.rest import Region, RegionInfo, BoundingBox, Concept

app = ClarifaiApp()

m.send_region_feedback(input_id='id2', url='https://developer.clarifai.com/static/
↪images/model-samples/celeb-001.jpg',
                       regions=[Region(region_info=RegionInfo(bbox=BoundingBox(top_
↪row=0.1,
                                                                               left_
↪col=0.2,
                                                                               bottom_
↪row=0.5,
```

(continues on next page)

(continued from previous page)

```
                                                                                right_
↪col=0.5)),
                                          concepts=[Concept(concept_id='people',␣
↪value=True),
                                                    Concept(concept_id='portrait',␣
↪value=True)])],
                        feedback_info=FeedbackInfo(output_id='OID',
                                                   session_id='SID',
                                                   end_user_id='UID',
                                                   event_type='annotation'))
```

Face detection model prediction

send feedback for celebrity model

```
from clarifai.rest import ClarifaiApp
from clarifai.rest import FeedbackInfo
from clarifai.rest import Region, RegionInfo, BoundingBox, Concept
from clarifai.rest import Face, FaceIdentity
from clarifai.rest import FaceAgeAppearance, FaceGenderAppearance,␣
↪FaceMulticulturalAppearance

app = ClarifaiApp()

#
# send feedback for celebrity model
#
m.send_region_feedback(input_id='id2', url='https://developer.clarifai.com/static/
↪images/model-samples/celeb-001.jpg',
                       regions=[Region(region_info=RegionInfo(bbox=BoundingBox(top_
↪row=0.1,
                                                                                left_
↪col=0.2,
                                                                                bottom_
↪row=0.5,
                                                                                right_
↪col=0.5)),
                                                 ␣
↪face=Face(identity=FaceIdentity([Concept(concept_id='celeb1', value=True)]))
                                                 )
                                ],
                       feedback_info=FeedbackInfo(output_id='OID',
                                                  session_id='SID',
                                                  end_user_id='UID',
                                                  event_type='annotation'))

#
# send feedback for age, gender, multicultural appearance
#

m.send_region_feedback(input_id='id2', url='https://developer.clarifai.com/static/
↪images/model-samples/celeb-001.jpg',
                       regions=[Region(region_info=RegionInfo(bbox=BoundingBox(top_
↪row=0.1,
                                                                                left_
↪col=0.2,
```

(continues on next page)

(continued from previous page)

```
                                                                                bottom_
↪row=0.5,
                                                                                right_
↪col=0.5)),
                                          face=Face(age_
↪appearance=FaceAgeAppearance([Concept(concept_id='20', value=True),
                                                                                    ␣
↪Concept(concept_id='30', value=False)
                                                                                  ]),
                                                  gender_
↪appearance=FaceGenderAppearance([Concept(concept_id='male', value=True)]),
                                                  multicultural_
↪appearance=FaceMulticulturalAppearance([Concept(concept_id='asian', value=True)])
                                                 )
                                         )
                                ],
                        feedback_info=FeedbackInfo(output_id='OID',
                                                   session_id='SID',
                                                   end_user_id='UID',
                                                   event_type='annotation'))
```

1.2.3 Upload Images

```
from clarifai.rest import ClarifaiApp

app = ClarifaiApp()

app.inputs.create_image_from_url(url='https://samples.clarifai.com/puppy.jpeg',␣
↪concepts=['my puppy'])
app.inputs.create_image_from_url(url='https://samples.clarifai.com/wedding.jpg', not_
↪concepts=['my puppy'])
```

1.2.4 Create a Model

Note: This assumes you follow through the tutorial and finished the "Upload Images" Otherwise you may not be able to create the model.

```
model = app.models.create(model_id="puppy", concepts=["my puppy"])
```

1.2.5 Train the Model

Note: This assumes you follow through the tutorial and finished the "Upload Images" and "Create a Model" to create a model. Otherwise you may not be able to train the model.

```
model.train()
```

1.2.6 Predict with Model

Note: This assumes you follow through the tutorial and finished the "Upload Images", "Create a Model", and "Train the Model". Otherwise you may not be able to make predictions with the model.

```
from clarifai.rest import ClarifaiApp

app = ClarifaiApp()

model = app.models.get('puppy')
model.predict_by_url('https://samples.clarifai.com/metro-north.jpg')
```

1.2.7 Instantiate an Image

```
from clarifai.rest import Image as ClImage

# make an image with an url
img = ClImage(url='https://samples.clarifai.com/dog1.jpeg')

# make an image with a filename
img = ClImage(filename='/tmp/user/dog.jpg')

# allow duplicate url
img = ClImage(url='https://samples.clarifai.com/dog1.jpeg', allow_dup_url=True)

# make an image with concepts
img = ClImage(url='https://samples.clarifai.com/dog1.jpeg', \
              concepts=['cat', 'animal'])

# make an image with metadata
img = ClImage(url='https://samples.clarifai.com/dog1.jpeg', \
              concepts=['cat', 'animal'], \
              metadata={'id':123,
                        'city':'New York'
                       })
```

1.2.8 Bulk Import Images

If you have a large amount of images, you may not want to upload them one by one by calling *app.inputs.create_image_from_url('https://samples.clarifai.com/dog1.jpeg')*

Instead you may want to use the bulk import API.

Note: The max number images per batch is 128. If you have more than 128 images to upload, you may want to chunk them into 128 or less, and bulk import them batch by batch.

In order to use this, you have to instantiate Image() objects from various sources.

```
from clarifai.rest import ClarifaiApp
from clarifai.rest import Image as ClImage
```

(continues on next page)

(continued from previous page)

```
3
4  # assume there are 100 urls in the list
5  images = []
6  for url in urls:
7    img = ClImage(url=url)
8    images.append(img)
9
10 app.inputs.bulk_create_images(images)
```

1.2.9 Search the Image

Note: This assumes you follow through the tutorial and finished the "Upload Images" Otherwise you may not be able to search

```
1  from clarifai.rest import ClarifaiApp
2
3  app = ClarifaiApp()
4
5  app.inputs.search_by_annotated_concepts(concept='my puppy')
6
7  app.inputs.search_by_predicted_concepts(concept='dog')
8
9  app.inputs.search_by_image(url='https://samples.clarifai.com/dog1.jpeg')
10
11 app.inputs.search_by_metadata(metadata={'key':'value'})
```

1.3 Basic Concepts

This page lists a few basic notions used in the Clarifai API.

1.3.1 Image

Image is a straightforward notion. It represents a picture in a digital format.

In the Clarifai API, an image could be represented by a url, a local filename, raw bytes of the image, or bytes encoded in base64. To construct a new Image object, use the Image() constructor.

1.3.2 Video

In the Clarifai API, Video is considered a sequence of frames where each frame represents one second of the video. This means that after running prediction, the models will return results for every second of the video.

Video is used similarly to Image. Video can be represented by a url, a local filename, raw bytes of the video, or bytes encoded in base64. To construct a new Video object, use the Video() constructor.

1.3.3 Input

Input is a more general notion; it can be either Image or Video. Input is used for uploading, prediction, search, etc.

Each Input has a unique ID and can be associated with Concepts.

1.3.4 Concept

Concept represents a word to associate with Inputs. Concept can be a concrete notion like "dog" or "cat", or a abstract notion like "love". All models of type *concept* return a set of Concepts in the prediction for the given Input. Each Concept has a unique ID, and a name (in the data science nomenclature also sometimes referred to as a label, or a tag).

1.3.5 Model

Model is a machine learning algorithm that takes Input, such as Image or Video, runs a prediction, and outputs the results. There are several types of models: Concept Model, Face Detection Model, Color Model, etc. They all return different types of results. For example, Concept Model returns associated Concepts for each Input, Face Detection Model returns the locations of faces, Color Model returns dominant colors, etc.

There are public (pre-defined) models you can use, and custom models that you train yourself with your own Inputs.

1.3.6 Custom models

Custom models must be of type *concept*.

When creating a custom model, you provide it with your Inputs and associated Concepts. After being trained, the model can predict what Concepts are associated with never-before-seen Inputs, together with probabilities for confidence of the prediction.

Models can also be evaluated for measuring their prediction capability.

1.3.7 Image Crop

Image crop is defined as a crop box within an image. We use this in visual search so user does not have to crop an image before the search.

We use percentage coordinates instead of pixel coordinates to specify the crop box.

A four-element-tuple represents a crop box, in (top_y, left_x, bottom_y, right_x) order.

So a (0.3, 0.2, 0.6, 0.4) represents a box horizontally spanning from 20%-40% and vertically spanning 30%-60% of the image, measured from the top left corner.

1.3.8 Workflow

Workflow enables you to run prediction on Inputs using several Models in one request.

Installation Guide Install Clarifai python library

Tutorial Getting started with Clarifai API using python client

Basic Concepts Getting familiar with basic concepts used in Clarifai API

contributing Contributing with a pull request

1.4 API Reference

This is the API Reference documentation extracted from the source code.

1.4.1 Application

class clarifai.rest.client.**ClarifaiApp**(*app_id=None*, *app_secret=None*, *base_url=None*, *api_key=None*, *quiet=True*, *log_level=None*)

Clarifai Application Object

This is the entry point of the Clarifai Client API. With authentication to an application, you can access all the models, concepts, and inputs in this application through the attributes of this class.

To access the models: use `app.models`
To access the inputs: use `app.inputs`
To access the concepts: use `app.concepts`

tag_files(*files*, *model_name='general-v1.3'*, *model_id=None*)

tag files on disk with user specified models by default tagged by 'general-v1.3' model

Parameters

- **files** – a list of local file names for tagging. The max length of the list is 128, which is the max batch size
- **model_name** – the model name to tag with. The default model is general model for general tagging purpose

Returns the JSON string from the predict call

Examples

```
>>> files = ['/tmp/metro-north.jpg',
>>>          '/tmp/dog2.jpeg']
>>> app.tag_urls(files)
```

tag_urls(*urls*, *model_name='general-v1.3'*, *model_id=None*)

tag urls with user specified models by default tagged by 'general-v1.3' model

Parameters

- **urls** – a list of URLs for tagging. The max length of the list is 128, which is the max batch size.
- **model_name** – the model name to tag with. The default model is general model for general tagging purpose

Returns the JSON string from the predict call

Examples

```
>>> urls = ['https://samples.clarifai.com/metro-north.jpg',
>>>         'https://samples.clarifai.com/dog2.jpeg']
>>> app.tag_urls(urls)
```

wait_until_inputs_delete_finish()
Block until a current inputs deletion operation finishes

The criteria for unblocking is 0 inputs returned from GET /inputs

Returns None

wait_until_inputs_upload_finish(*max_wait=666666*)
Block until the inputs upload finishes

The criteria for unblocking is 0 "to_process" inputs from GET /inputs/status

Returns None

wait_until_models_delete_finish()
Block until the inputs deletion finishes

The criteria for unblocking is 0 models returned from GET /models

Returns None

1.4.2 Concepts

class clarifai.rest.client.**Concepts**(*api*)

bulk_create(*concept_ids*, *concept_names=None*)
Bulk create concepts

When the concept name is not set, it will be set as the same as concept ID.

Parameters

- **concept_ids** – a list of concept IDs, in sequence
- **concept_names** – a list of corresponding concept names, in the same sequence

Returns A list of Concept objects

Examples

```
>>> app.concepts.bulk_create(['id1', 'id2'], ['cute cat', 'cute dog'])
```

bulk_update(*concept_ids*, *concept_names*, *action='overwrite'*)
Patch multiple concepts

Parameters

- **concept_ids** – a list of concept IDs, in sequence
- **concept_names** – a list of corresponding concept names, in the same sequence
- **action** – the type of update

Returns the new Concept object

Examples

```
>>> app.concepts.bulk_update(concept_ids=['myid1', 'myid2'],
>>>                          concept_names=['name2', 'name3'])
```

create(*concept_id*, *concept_name=None*)
Create a new concept

Parameters

- **concept_id** – concept ID, the unique identifier of the concept
- **concept_name** – name of the concept. If name is not specified, it will be set to the same as the concept ID

Returns the new Concept object

get(*concept_id*)
Get a concept by id

Parameters **concept_id** – concept ID, the unique identifier of the concept

Returns If found, return the Concept object. Otherwise, return None

Examples

```
>>> app.concepts.get('id')
```

get_all()
Get all concepts associated with the application

Returns all concepts in a generator function

get_by_page(*page=1*, *per_page=20*)
get concept with pagination

Parameters

- **page** – page number
- **per_page** – number of concepts to retrieve per page

Returns a list of Concept objects

Examples

```
>>> for concept in app.concepts.get_by_page(2, 10):
>>>     print(concept.concept_id)
```

search(*term*, *lang=None*)
search concepts by concept name with wildcards

Parameters

- **term** – search term with wildcards allowed
- **lang** – language to search, if none is specified the default for the application will be used

Returns a generator function with all concepts pertaining to the search term

Examples

```
>>> app.concepts.search('cat')
>>> # search for Chinese label name
>>> app.concepts.search(u'*', lang='zh')
```

update(*concept_id*, *concept_name*, *action='overwrite'*)
Patch concept

Parameters

- **concept_id** – id of the concept
- **concept_name** – the new name for the concept

Returns the new Concept object

Examples

```
>>> app.concepts.update(concept_id='myid1', concept_name='new_concept_name2')
```

1.4.3 Inputs

class clarifai.rest.client.**Inputs**(*api*)

add_concepts(*input_id*, *concepts*, *not_concepts*)
Add concepts for one input

This is just an alias of *merge_concepts* for easier understanding when you try to add some new concepts to an image

Parameters

- **input_id** – the unique ID of the input
- **concepts** – the list of concepts
- **not_concepts** – the list of negative concepts

Returns an Input object

Examples

```
>>> app.inputs.add_concepts('id', ['cat', 'kitty'], ['dog'])
```

bulk_create_images(*images*)
Create images in bulk

Parameters **images** – a list of Image objects

Returns a list of the Image objects that were just created

Examples

```
>>> img1 = Image(url="", concepts=['cat', 'kitty'])
>>> img2 = Image(url="", concepts=['dog'], not_concepts=['cat'])
>>> app.inputs.bulk_create_images([img1, img2])
```

bulk_delete_concepts(*input_ids*, *concept_lists*)
bulk delete concepts from a list of input ids

Parameters

- **input_ids** – a list of input IDs
- **concept_lists** – a list of concept lists, each one corresponding to a listed input ID and
- **with concepts to be deleted from that input**(*filled*) –

Returns an Input object

Examples

```
>>> app.inputs.bulk_delete_concepts(['id'], [['cat', 'dog']])
```

bulk_merge_concepts(*input_ids*, *concept_lists*)
bulk merge concepts from a list of input ids

Parameters

- **input_ids** – a list of input IDs
- **concept_lists** – a list of concept lists, each one corresponding to a listed input ID and
- **with concepts to be added to that input**(*filled*) –

Returns an Input object

Examples

```
>>> app.inputs.bulk_merge_concepts('id', [[('cat',True), ('dog',False)]])
```

bulk_update(*images*, *action='merge'*)
Update the input update the information of an input/image

Parameters

- **images** – a list of Image objects that have concepts, metadata, etc.
- **action** – one of [‘merge’, ‘overwrite’]

  ’merge’ is to append the info onto the exising info, for either concept or metadata

  ’overwrite’ is to overwrite the existing metadata and concepts with the existing ones

Returns an Image object

Examples

```
>>> new_img1 = Image(image_id="abc1", concepts=['c1', 'c2'], not_concepts=['c3
↪'],
>>>                  metadata={'key':'val'})
>>> new_img2 = Image(image_id="abc2", concepts=['c1', 'c2'], not_concepts=['c3
↪'],
>>>                  metadata={'key':'val'})
>>> app.inputs.update([new_img1, new_img2], action='overwrite')
```

check_status()
check the input upload status

Returns InputCounts object

Examples

```
>>> status = app.inputs.check_status()
>>> print(status.code)
>>> print(status.description)
```

create_image(*image*)
create an image from Image object

Parameters **image** – a Clarifai Image object

Returns the Image object that just got created and uploaded

Examples

```
>>> app.inputs.create_image(Image(url='https://samples.clarifai.com/metro-
↪north.jpg'))
```

create_image_from_base64(*base64_bytes*, *image_id=None*, *concepts=None*, *not_concepts=None*, *crop=None*, *metadata=None*, *geo=None*, *allow_duplicate_url=False*)
create an image by base64 bytes

Parameters

- **base64_bytes** – base64 encoded image bytes
- **image_id** – ID of the image
- **concepts** – a list of concept names this image is associated with
- **not_concepts** – a list of concept names this image is not associated with
- **crop** – crop information, with four corner coordinates
- **metadata** – meta data with a dictionary
- **geo** – geo info with a dictionary
- **allow_duplicate_url** – True or False, the flag to allow a duplicate url to be imported

Returns the Image object that just got created and uploaded

Examples

```
>>> app.inputs.create_image_bytes(base64_bytes="base64 encoded image bytes...
↪")
```

create_image_from_bytes(*img_bytes*, *image_id=None*, *concepts=None*, *not_concepts=None*, *crop=None*, *metadata=None*, *geo=None*, *allow_duplicate_url=False*)

create an image by image bytes

Parameters

- **img_bytes** – raw bytes of an image
- **image_id** – ID of the image
- **concepts** – a list of concept names this image is associated with
- **not_concepts** – a list of concept names this image is not associated with
- **crop** – crop information, with four corner coordinates
- **metadata** – meta data with a dictionary
- **geo** – geo info with a dictionary
- **allow_duplicate_url** – True or False, the flag to allow a duplicate url to be imported

Returns the Image object that just got created and uploaded

Examples

```
>>> app.inputs.create_image_bytes(img_bytes="raw image bytes...")
```

create_image_from_filename(*filename*, *image_id=None*, *concepts=None*, *not_concepts=None*, *crop=None*, *metadata=None*, *geo=None*, *allow_duplicate_url=False*)

create an image by local filename

Parameters

- **filename** – local filename
- **image_id** – ID of the image
- **concepts** – a list of concept names this image is associated with
- **not_concepts** – a list of concept names this image is not associated with
- **crop** – crop information, with four corner coordinates
- **metadata** – meta data with a dictionary
- **geo** – geo info with a dictionary
- **allow_duplicate_url** – True or False, the flag to allow a duplicate url to be imported

Returns the Image object that just got created and uploaded

Examples

```
>>> app.inputs.create_image_filename(filename="a.jpeg")
```

create_image_from_url(*url*, *image_id=None*, *concepts=None*, *not_concepts=None*, *crop=None*, *metadata=None*, *geo=None*, *allow_duplicate_url=False*)

create an image from Image url

Parameters

- **url** – image url
- **image_id** – ID of the image
- **concepts** – a list of concept names this image is associated with
- **not_concepts** – a list of concept names this image is not associated with
- **crop** – crop information, with four corner coordinates
- **metadata** – meta data with a dictionary
- **geo** – geo info with a dictionary
- **allow_duplicate_url** – True or False, the flag to allow a duplicate url to be imported

Returns the Image object that just got created and uploaded

Examples

```
>>> app.inputs.create_image_from_url(url='https://samples.clarifai.com/metro-
↪north.jpg')
>>>
>>> # create image with geo point
>>> app.inputs.create_image_from_url(url='https://samples.clarifai.com/metro-
↪north.jpg',
>>>                                  geo=Geo(geo_point=GeoPoint(22.22, 44.44))
```

delete(*input_id*)

delete an input with input ID

Parameters **input_id** – the unique input ID

Returns ApiStatus object

Examples

```
>>> ret = app.inputs.delete('id1')
>>> print(ret.code)
```

delete_all()

delete all inputs from the application

delete_concepts(*input_id*, *concepts*)

delete concepts from an input/image

Parameters

- **input_id** – unique ID of the input
- **concepts** – a list of concept names

Returns an Image object

get (*input_id*)
get an Input object by input ID

Parameters **input_id** – the unique identifier of the input

Returns an Image/Input object

Examples

```
>>> image = app.inputs.get('id1')
>>> print(image.input_id)
```

get_all (*ignore_error=False*)
Get all inputs

Parameters **ignore_error** – ignore errored inputs. For example some images may fail to be imported due to bad url

Returns a generator function that yields Input objects

Examples

```
>>> for image in app.inputs.get_all():
>>>     print(image.input_id)
```

get_by_page (*page=1*, *per_page=20*, *ignore_error=False*)
Get inputs with pagination

Parameters

- **page** – page number
- **per_page** – number of inputs to retrieve per page
- **ignore_error** – ignore errored inputs. For example some images may fail to be imported due to bad url

Returns a list of Input objects

Examples

```
>>> for image in app.inputs.get_by_page(2, 10):
>>>     print(image.input_id)
```

merge_concepts (*input_id*, *concepts*, *not_concepts*, *overwrite=False*)
Merge concepts for one input

Parameters

- **input_id** – the unique ID of the input
- **concepts** – the list of concepts

- **not_concepts** – the list of negative concepts
- **overwrite** – if True, this operation will replace the previous concepts. If False,
- **will append them.** (*it*) –

Returns an Input object

Examples

```
>>> app.inputs.merge_concepts('id', ['cat', 'kitty'], ['dog'])
```

merge_metadata(*input_id*, *metadata*)
merge metadata for the image

This is to merge/update the metadata of the given image

Parameters

- **input_id** – the unique ID of the input
- **metadata** – the metadata dictionary

Examples

```
>>> # merge the metadata
>>> # metadata will be appended to the existing key/value pairs
>>> app.inputs.merge_metadata('id', {'key1':'value1', 'key2':'value2'})
```

search(*qb*, *page=1*, *per_page=20*, *raw=False*)
search with a clarifai image query builder

WARNING: this is the advanced search function. You will need to build a query builder in order to use this.

There are a few simple search functions: search_by_annotated_concepts() search_by_predicted_concepts() search_by_image() search_by_metadata()

Parameters

- **qb** – clarifai query builder
- **raw** – raw result indicator

Returns a list of Input/Image object

search_by_annotated_concepts(*concept=None*, *concepts=None*, *value=True*, *values=None*, *concept_id=None*, *concept_ids=None*, *page=1*, *per_page=20*, *raw=False*)
search using the concepts the user has manually specified

Parameters

- **concept** – concept name to search
- **concepts** – a list of concept name to search
- **concept_id** – concept IDs to search
- **concept_ids** – a list of concept IDs to search

- **value** – whether the concept should be a positive tag or negative
- **values** – the list of values corresponding to the concepts
- **page** – page number
- **per_page** – number of images to return per page
- **raw** – raw result indicator

Returns a list of Image objects

Examples

```
>>> app.inputs.search_by_annotated_concepts(concept='cat')
```

search_by_geo(*geo_point=None*, *geo_limit=None*, *geo_box=None*, *page=1*, *per_page=20*, *raw=False*)

search by geo point and geo limit

Parameters

- **geo_point** – A GeoPoint object, which represents the (longitude, latitude) of a location
- **geo_limit** – A GeoLimit object, which represents a range to a GeoPoint
- **geo_box** – A GeoBox object, which represents a box area
- **page** – page number
- **per_page** – number of images to return per page
- **raw** – raw result indicator

Returns a list of Image objects

Examples

```
>>> app.inputs.search_by_geo(GeoPoint(30, 40), GeoLimit("mile", 10))
```

search_by_image(*image_id=None*, *image=None*, *url=None*, *imgbytes=None*, *base64bytes=None*, *fileobj=None*, *filename=None*, *crop=None*, *page=1*, *per_page=20*, *raw=False*)

Search for visually similar images

By passing image_id, raw image bytes, base64 encoded bytes, image file io stream, image filename, or Clarifai Image object, you can use the visual search power of the Clarifai API.

You can specify a crop of the image to search over

Parameters

- **image_id** – unique ID of the image for search
- **image** – Image object for search
- **imgbytes** – raw image bytes for search
- **base64bytes** – base63 encoded image bytes
- **fileobj** – file io stream, like open(file)
- **filename** – filename on local filesystem

- **crop** – crop of the image as a list of four floats representing the corner coordinates
- **page** – page number
- **per_page** – number of images returned per page
- **raw** – raw result indicator

Returns a list of Image object

Examples

```
>>> # search by image url
>>> app.inputs.search_by_image(url='http://blabla')
>>> # search by local filename
>>> app.inputs.search_by_image(filename='bla')
>>> # search by raw image bytes
>>> app.inputs.search_by_image(imgbytes='data')
>>> # search by base64 encoded image bytes
>>> app.inputs.search_by_image(base64bytes='data')
>>> # search by file stream io
>>> app.inputs.search_by_image(fileobj=open('file'))
```

search_by_metadata(*metadata*, *page=1*, *per_page=20*, *raw=False*)
search by meta data of the image rather than concept

Parameters

- **metadata** – a dictionary for meta data search. The dictionary could be a simple one with only one key and value, Or a nested dictionary with multi levels.
- **page** – page number
- **per_page** – the number of images to return per page
- **raw** – raw result indicator

Returns a list of Image objects

Examples

```
>>> app.inputs.search_by_metadata(metadata={'name':'bla'})
>>> app.inputs.search_by_metadata(metadata={'my_class1': { 'name' : 'bla' }})
```

search_by_original_url(*url*, *page=1*, *per_page=20*, *raw=False*)
search by the original url of the uploaded images

Parameters

- **url** – url of the image
- **page** – page number
- **per_page** – the number of images to return per page
- **raw** – raw result indicator

Returns a list of Image objects

Examples

```
>>> app.inputs.search_by_original_url(url='http://bla')
```

search_by_predicted_concepts(*concept=None*, *concepts=None*, *value=True*, *values=None*, *concept_id=None*, *concept_ids=None*, *page=1*, *per_page=20*, *lang=None*, *raw=False*)

search over the predicted concepts

Parameters

- **concept** – concept name to search
- **concepts** – a list of concept names to search
- **concept_id** – concept id to search
- **concept_ids** – a list of concept ids to search
- **value** – whether the concept should be a positive tag or negative
- **values** – the list of values corresponding to the concepts
- **page** – page number
- **per_page** – number of images to return per page
- **lang** – language to search over for translated concepts
- **raw** – raw result indicator

Returns a list of Image objects

Examples

```
>>> app.inputs.search_by_predicted_concepts(concept='cat')
>>> # search over simplified Chinese label
>>> app.inputs.search_by_predicted_concepts(concept=u'', lang='zh')
```

send_search_feedback(*input_id*, *feedback_info=None*)

Send feedback for search

Parameters **input_id** – unique identifier for the input

Returns None

update(*image*, *action='merge'*)

Update the information of an input/image

Parameters

- **image** – an Image object that has concepts, metadata, etc.
- **action** – one of ['merge', 'overwrite']

  'merge' is to append the info onto the existing info, for either concept or metadata

  'overwrite' is to overwrite the existing metadata and concepts with the existing ones

Returns an Image object

Examples

```
>>> new_img = Image(image_id="abc", concepts=['c1', 'c2'], not_concepts=['c3
↪'],
>>>                 metadata={'key':'val'})
>>> app.inputs.update(new_img, action='overwrite')
```

1.4.4 Models

class clarifai.rest.client.**Models**(*api*, *solutions*)

bulk_delete(*model_ids*)
Delete multiple models.

Parameters **model_ids** – a list of unique IDs of the models to delete

Returns the raw JSON response from the server

Examples

```
>>> app.models.delete_models(['model_id1', 'model_id2'])
```

clear_model_cache()
clear model_name -> model_id cache

WARNING: This is an internal function, user should not call this

We cache model_name to model_id mapping for API efficiency. The first time you call a models.get() by name, the name to ID mapping is saved so next time there is no query. Then user does not have to query the model ID every time when they want to work on it.

create(*model_id*, *model_name=None*, *concepts=None*, *concepts_mutually_exclusive=False*, *closed_environment=False*, *hyper_parameters=None*)
Create a new model

Parameters

- **model_id** – ID of the model
- **model_name** – optional name of the model
- **concepts** – optional concepts to be associated with this model
- **concepts_mutually_exclusive** – True or False, whether concepts are mutually exclusive
- **closed_environment** – True or False, whether to use negatives for prediction
- **hyper_parameters** – hyper parameters for the model, with a json object

Returns Model object

Examples

```
>>> # create a model with no concepts
>>> app.models.create('my_model1')
>>> # create a model with a few concepts
>>> app.models.create('my_model2', concepts=['bird', 'fish'])
>>> # create a model with closed environment
>>> app.models.create('my_model3', closed_environment=True)
```

delete(*model_id*, *version_id=None*)

delete the model, or a specific version of the model

Without model version id specified, all the versions associated with this model will be deleted as well.

With model version id specified, it will delete a particular model version from the model

Parameters

- **model_id** – the unique ID of the model
- **version_id** – the unique ID of the model version

Returns the raw JSON response from the server

Examples

```
>>> # delete a model
>>> app.models.delete('model_id1')
>>> # delete a model version
>>> app.models.delete('model_id1', version_id='version1')
```

delete_all()

Delete all models and the versions associated with each one

After this operation, you will have no models in the application

Returns the raw JSON response from the server

Examples

```
>>> app.models.delete_all()
```

get(*model_name=None*, *model_id=None*, *model_type=None*)

Get a model, by ID or name

Parameters

- **model_name** – name of the model
- **model_id** – unique identifier of the model
- **model_type** – type of the model

Returns the Model object

Examples

```
>>> # get general-v1.3 model
>>> app.models.get('general-v1.3')
```

get_all (*public_only=False*, *private_only=False*)
Get all models in the application

Parameters

- **public_only** – only yield public models
- **private_only** – only yield private models that tie to your own account

Returns a generator function that yields Model objects

Examples

```
>>> for model in app.models.get_all():
>>>     print(model.model_name)
```

get_by_page (*public_only=False*, *private_only=False*, *page=1*, *per_page=20*)
get paginated models from the application

When the number of models gets high, you may want to get the paginated results from all the models

Parameters

- **public_only** – only yield public models
- **private_only** – only yield private models that tie to your own account
- **page** – page number
- **per_page** – number of models returned in one page

Returns a list of Model objects

Examples

```
>>> models = app.models.get_by_page(2, 20)
```

init_model_cache ()
Initialize the model cache for the public models

This will go through all public models and cache them

Returns JSON object containing the name, type, and id of all cached models

search (*model_name*, *model_type=None*)
Search the model by name and optionally type. Default is to search concept models only. All the custom model trained are concept models.

Parameters

- **model_name** – name of the model. name is not unique.
- **model_type** – default to None, equivalent to wildcards search

Returns a list of Model objects or None

Examples

```
>>> # search for general-v1.3 models
>>> app.models.search('general-v1.3')
>>>
>>> # search for color model
>>> app.models.search('color', model_type='color')
>>>
>>> # search for face model
>>> app.models.search('face-v1.3', model_type='facedetect')
```

1.4.5 Model

class clarifai.rest.client.**Model**(*api*, *item=None*, *model_id=None*, *solutions=None*)

add_concepts(*concept_ids*)
merge concepts into a model

This is just an alias of *merge_concepts*, for easier understanding of adding new concepts to the model without overwritting them.

Parameters **concept_ids** – a list of concept IDs

Returns the Model object

Examples

```
>>> model = self.app.models.get('model_id')
>>> model.add_concepts(['cat', 'dog'])
```

delete_concepts(*concept_ids*)
delete concepts from a model

Parameters **concept_ids** – a list of concept IDs to be removed

Returns the Model object

Examples

```
>>> model = self.app.models.get('model_id')
>>> model.delete_concepts(['cat', 'dog'])
```

delete_version(*version_id*)
delete model version by version_id

Parameters **version_id** – version id of the model version

Returns the JSON response

Examples

```
>>> model = self.app.models.get('model_id')
>>> model.delete_version('model_version_id')
```

evaluate()
run model evaluation

Returns the model version data with evaluation metrics in JSON format

get_concept_ids()
get concepts IDs associated with the model

Returns a list of concept IDs

Examples

```
>>> ids = model.get_concept_ids()
```

get_info(*verbose=False*)
get model info, with or without the concepts associated with the model.

Parameters **verbose** – default is False. True will yield output_info, with concepts of the model

Returns raw json of the response

Examples

```
>>> # with basic model info
>>> model.get_info()
>>> # model info with concepts
>>> model.get_info(verbose=True)
```

get_inputs(*version_id=None*, *page=1*, *per_page=20*)
Get all the inputs from the model or a specific model version. Without specifying a model version id, this will yield all inputs

Parameters

- **version_id** – model version id
- **page** – page number
- **per_page** – number of inputs to return for each page

Returns A list of Input objects

get_version(*version_id*)
get model version info for a particular version

Parameters **version_id** – version id of the model version

Returns the JSON response

Examples

```
>>> model = self.app.models.get('model_id')
>>> model.get_version('model_version_id')
```

list_versions()
list all model versions

Returns the JSON response

Examples

```
>>> model = self.app.models.get('model_id')
>>> model.list_versions()
```

merge_concepts(*concept_ids*, *overwrite=False*)
merge concepts in a model

When overwrite is False, if the concept does not exist in the model it will be appended. Otherwise, the original one will be kept.

Parameters

- **concept_ids** – a list of concept id
- **overwrite** – True or False. If True, the existing concepts will be replaced

Returns the Model object

predict(*inputs*, *model_output_info=None*)
predict with multiple images

Parameters **inputs** – a list of Image objects

Returns the prediction of the model in JSON format

predict_by_base64(*base64_bytes*, *lang=None*, *is_video=False*, *min_value=None*, *max_concepts=None*, *select_concepts=None*)
predict a model with base64 encoded image bytes

Parameters

- **base64_bytes** – base64 encoded image bytes
- **lang** – language to predict, if the translation is available
- **is_video** – whether this is a video
- **min_value** – threshold to cut the predictions, 0-1.0
- **max_concepts** – max concepts to keep in the predictions, 0-200
- **select_concepts** – a list of concepts that are selected to be exposed

Returns the prediction of the model in JSON format

predict_by_bytes(*raw_bytes*, *lang=None*, *is_video=False*, *min_value=None*, *max_concepts=None*, *select_concepts=None*)
predict a model with image raw bytes

Parameters

- **raw_bytes** – raw bytes of an image

- **lang** – language to predict, if the translation is available
- **is_video** – whether this is a video
- **min_value** – threshold to cut the predictions, 0-1.0
- **max_concepts** – max concepts to keep in the predictions, 0-200
- **select_concepts** – a list of concepts that are selected to be exposed

Returns the prediction of the model in JSON format

predict_by_filename(*filename*, *lang=None*, *is_video=False*, *min_value=None*, *max_concepts=None*, *select_concepts=None*)

predict a model with a local filename

Parameters

- **filename** – filename on local filesystem
- **lang** – language to predict, if the translation is available
- **is_video** – whether this is a video
- **min_value** – threshold to cut the predictions, 0-1.0
- **max_concepts** – max concepts to keep in the predictions, 0-200
- **select_concepts** – a list of concepts that are selected to be exposed

Returns the prediction of the model in JSON format

predict_by_url(*url*, *lang=None*, *is_video=False*, *min_value=None*, *max_concepts=None*, *select_concepts=None*)

predict a model with url

Parameters

- **url** – publicly accessible url of an image
- **lang** – language to predict, if the translation is available
- **is_video** – whether this is a video
- **min_value** – threshold to cut the predictions, 0-1.0
- **max_concepts** – max concepts to keep in the predictions, 0-200
- **select_concepts** – a list of concepts that are selected to be exposed

Returns the prediction of the model in JSON format

send_concept_feedback(*input_id*, *url*, *concepts=None*, *not_concepts=None*, *feedback_info=None*)

Send feedback for this model

Parameters

- **input_id** – input id for the feedback
- **url** – the url of the input
- **concepts** – concepts that are present
- **not_concepts** – concepts that aren't present
- **feedback_info** – feedback info

Returns None

send_region_feedback(*input_id*, *url*, *concepts=None*, *not_concepts=None*, *regions=None*, *feedback_info=None*)

Send feedback for this model

Parameters

- **input_id** – input id for the feedback
- **url** – the input url

Returns None

train(*sync=True*, *timeout=60*)

train the model in synchronous or asynchronous mode. Synchronous will block until the model is trained, async will not.

Parameters

- **sync** – indicating synchronous or asynchronous, default is True
- **timeout** – Used when sync=True. Num. of seconds we should wait for training to complete.

Returns the Model object

update(*action='merge'*, *model_name=None*, *concepts_mutually_exclusive=None*, *closed_environment=None*, *concept_ids=None*)

Update the model attributes. The name of the model, list of concepts, and the attributes `concepts_mutually_exclusive` and `closed_environment` can be changed. Note this is a overwriting change. For a valid call, at least one or more attributes should be specified. Otherwise the call will be just skipped without error.

Parameters

- **action** – the way to patch the model: ['merge', 'remove', 'overwrite']
- **model_name** – name of the model
- **concepts_mutually_exclusive** – whether the concepts are mutually exclusive
- **closed_environment** – whether negative concepts should be taken into account during training
- **concept_ids** – a list of concept ids

Returns the Model object

Examples

```
>>> model = self.app.models.get('model_id')
>>> model.update(model_name="new_model_name")
>>> model.update(concepts_mutually_exclusive=False)
>>> model.update(closed_environment=True)
>>> model.update(concept_ids=["bird", "hurd"])
>>> model.update(concepts_mutually_exclusive=True, concept_ids=["bird", "hurd
↪"])
```

1.4.6 Search Syntax

class clarifai.rest.client.**SearchTerm**

Clarifai search term interface. This is the base class for InputSearchTerm and OutputSearchTerm

It is used to build SearchQueryBuilder

class clarifai.rest.client.**InputSearchTerm**(*url=None*, *input_id=None*, *concept=None*, *concept_id=None*, *value=True*, *metadata=None*, *geo=None*)

Clarifai Input Search Term for an image search. For input search, you can specify search terms for url string match, input_id string match, concept string match, concept_id string match, and geographic information. Value indicates whether the concept search is a NOT search

Examples

```
>>> # search for url, string match
>>> InputSearchTerm(url='http://blabla')
>>> # search for input ID, string match
>>> InputSearchTerm(input_id='site1_bla')
>>> # search for annotated concept
>>> InputSearchTerm(concept='tag1')
>>> # search for not the annotated concept
>>> InputSearchTerm(concept='tag1', value=False)
>>> # search for metadata
>>> InputSearchTerm(metadata={'key':'value'})
>>> # search for geo
>>> InputSearchTerm(geo=Geo(geo_point=GeoPoint(-40, 30),
>>>                 geo_limit=GeoLimit('withinMiles', 10)))
```

class clarifai.rest.client.**OutputSearchTerm**(*url=None*, *base64=None*, *input_id=None*, *concept=None*, *concept_id=None*, *value=True*, *crop=None*)

Clarifai Output Search Term for image search. For output search, you can specify search term for url, base64, and input_id for visual search, or specify concept and concept_id for string match. Value indicates whether the concept search is a NOT search

Examples

```
>>> # search for visual similarity from url
>>> OutputSearchTerm(url='http://blabla')
>>> # search for visual similarity from base64 encoded image
>>> OutputSearchTerm(base64='sdfds')
>>> # search for visual similarity from input id
>>> OutputSearchTerm(input_id='site1_bla')
>>> # search for predicted concept
>>> OutputSearchTerm(concept='tag1')
>>> # search for not the predicted concept
>>> OutputSearchTerm(concept='tag1', value=False)
```

class clarifai.rest.client.**SearchQueryBuilder**(*language=None*)

Clarifai Image Search Query Builder

This builder is for advanced search use ONLY.

If you are looking for simple concept search, or simple image similarity search, you should use one of the existing functions `search_by_annotated_concepts`, `search_by_predicted_concepts`, `search_by_image` or `search_by_metadata`

Currently the query builder only supports a list of query terms with AND. InputSearchTerm and OutputSearchTerm are the only terms supported by the query builder

Examples

```
>>> qb = SearchQueryBuilder()
>>> qb.add_term(term1)
>>> qb.add_term(term2)
>>>
>>> app.inputs.search(qb)
>>>
>>> # for search over translated output concepts
>>> qb = SearchQueryBuilder(language='zh')
>>> qb.add_term(term1)
>>> qb.add_term(term2)
>>>
>>> app.inputs.search(qb)
```

add_term(*term*)
add a search term to the query. This can search by input or by output. Construct the term argument with an InputSearchTerm or OutputSearchTerm object.

dict()
construct the raw query for the RESTful API

1.4.7 Exceptions

class clarifai.rest.client.**ApiError**(*resource*, *params*, *method*, *response=None*)
API Server error

class clarifai.rest.client.**ApiClientError**
API Client Error

class clarifai.rest.client.**UserError**
User Error

Index