

---

# **clarifai Documentation**

***Release 2.0.30***

**Clarifai**

**Jul 19, 2017**



<b>1</b>	<b>First steps</b>	<b>1</b>
1.1	Installation Guide . . . . .	1
1.2	Tutorial . . . . .	2
1.3	Basic Concepts . . . . .	8
1.4	API Reference . . . . .	9



## Installation Guide

**Note:** Generally, the python client works with Python 2.x and Python 3.x. But it is only tested against 2.7 and 3.5. Feel free to report BUGs if you encounter on other versions.

### 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==2.0.30
```

You can also install from git using latest source:

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

### Configuration

The client uses CLARIFAI\_APP\_ID and CLARIFAI\_APP\_SECRET for authentication and token generation. Each application you create uses its own unique ID and secret to authenticate requests. The client will use the authentication information passed to it by one of three methods with the following precedence order:

1. Passed in to the constructor through the `app_id` and `app_secret` parameters.
2. Set as the CLARIFAI\_APP\_ID and CLARIFAI\_APP\_SECRET environment variables
3. Placed in the `.clarifai/config` file using the command below.

You can get these values from <https://developer.clarifai.com/account/applications> and then run:

```
$ clarifai config
CLARIFAI_APP_ID: []: *****YQEd
CLARIFAI_APP_SECRET: []: *****gCqT
```

If you do not see any error message after this, you are all set and can proceed with using the client.

## Windows Users

For Windows users, you may fail running the `clarifai config` 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 nonexecutable. In order to run the command, you may want to launch it with the python interpreter.

```
C:\Python27>python.exe Scripts\clarifai config
CLARIFAI_APP_ID: []: *****YQEd
CLARIFAI_APP_SECRET: []: *****gCqT
```

## AWS Lambda Users

For AWS Lambda users, in order to use the library correctly, you are recommended to set two environmental variables `CLARIFAI_APP_ID` and `CLARIFAI_APP_SECRET` in the lambda function configuration, or hardcode the `APP_ID` and `APP_SECRET` in the API instantiation.

## 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.

### 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')

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')
```

## 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_
right_
↳row=0.5,
↳col=0.5)),
        concepts=[Concept (concept_id='people',
        Concept (concept_id='portrait',
↳value=True),
↳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))),
        ↳
        ],
        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,

```

```
↪row=0.5,
↪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'))
```

## Upload Images

```
1 from clarifai.rest import ClarifaiApp
2
3 app = ClarifaiApp()
4
5 app.inputs.create_image_from_url(url='https://samples.clarifai.com/puppy.jpeg',
↪concepts=['my puppy'])
6 app.inputs.create_image_from_url(url='https://samples.clarifai.com/wedding.jpg', not_
↪concepts=['my puppy'])
```

## 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.

---

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

## 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.

---

```
1 model.train()
```

## 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.

```

1 from clarifai.rest import ClarifaiApp
2
3 app = ClarifaiApp()
4
5 model = app.models.get('puppy')
6 model.predict_by_url('https://samples.clarifai.com/metro-north.jpg')

```

## Instantiate an Image

```

1 from clarifai.rest import Image as CImage
2
3 # make an image with an url
4 img = CImage(url='https://samples.clarifai.com/dog1.jpeg')
5
6 # make an image with a filename
7 img = CImage(filename='/tmp/user/dog.jpg')
8
9 # allow duplicate url
10 img = CImage(url='https://samples.clarifai.com/dog1.jpeg', allow_dup_url=True)
11
12 # make an image with concepts
13 img = CImage(url='https://samples.clarifai.com/dog1.jpeg', \
14             concepts=['cat', 'animal'])
15
16 # make an image with metadata
17 img = CImage(url='https://samples.clarifai.com/dog1.jpeg', \
18             concepts=['cat', 'animal'], \
19             metadata={'id':123,
20                     'city':'New York'
21                     })

```

## 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.

```

1 from clarifai.rest import ClarifaiApp
2 from clarifai.rest import Image as CImage
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)
```

## 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'})
```

## Basic Concepts

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

### Image

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

In Clarifai API, an image could be represented by an url, a local filename, an opened io stream, raw bytes of the image, or bytes encoded in base64.

There is a Image() class to construct an Image object in Clarifai API.

Along with the image bytes, an image could also be associated with an unique ID, as well as one or more concepts indicating what it is and what is it not.

Image object could be used in uploading, prediction, as well as search.

### Input

Input is a generalized image, because input could be image, or video in the near future. In the current API, we use Image and Input interchangeably because Image is the only Input type the API supports.

Similarly, input could be associated with ID and concepts. And Input is used in uploading, prediction, and search.

## Concept

Concept represents a class to associate with images. It could be a concrete concept like “dog” or “cat”, or a abstract concept like “love”. It is the output of the concept Models. Clarifai Concept has an unique ID, and a name. Sometimes the concept name is also referred to label or tag for the predictions. They are interchangeably used in some places.

## Model

Model is a machine learning algorithm that takes input, such as image or video, and outputs some results for prediction. For custom training, the models trained are all concept models. The output of a concept model is a list of concepts in prediction, associated with probabilities for confidence of the prediction.

## Image Crop

Image crop is the definition of the 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.

*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

## API Reference

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

## Application

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

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`

**check\_upgrade()**

Check for a client upgrade. If the client has been installed for more than one week, the check will be triggered. If the newer version is available, a prompt message will pop up as a warning message in STDERR. The API call will not be paused or interrupted.

**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** – 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** – 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

**Parameters** **void** –

**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

**Parameters** `void` –

**Returns** `None`

**`wait_until_models_delete_finish()`**

Block until the inputs deletion finishes

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

**Parameters** `void` –

**Returns** `None`

## 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

**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

**Parameters** **void** –

**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')
```

## 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 filled with concepts to be deleted from that input

**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 filled with concepts to be added to that input

**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 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_img1 = Image(image_id="abc1", concepts=['c1', 'c2'], not_concepts=['c3', 'c4'], metadata={'key': 'val'})
>>> new_img2 = Image(image_id="abc2", concepts=['c1', 'c2'], not_concepts=['c3', 'c4'], metadata={'key': 'val'})
>>> app.inputs.update([new_img1, new_img2], action='overwrite')
```

**check\_status** ()

check the input upload status

**Parameters** Void –

**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('idl')
>>> 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('idl')
>>> 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
```

**get\_outputs** (*input\_id*)

get the output predictions for a particular input

**Parameters** **input\_id** – the unique identifier of the input

**Returns** the input with the output predictions

**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, it will append them.

**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'})
```

**merge\_outputs\_concepts** (*input\_id, concept\_ids*)

Merge new concepts into the outputs predictions. The concept ids must be present in your app

### Parameters

- **input\_id** – the unique identifier of the input
- **concept\_ids** – the list of concept ids to be merged

**Returns** the patched input in JSON object

**remove\_outputs\_concepts** (*input\_id, concept\_ids*)

Remove concepts from the outputs predictions. The concept ids must be present in your app

### Parameters

- **input\_id** – the unique identifier of the input
- **concept\_ids** – the list of concept ids to be removed

**Returns** the patched input in JSON object

**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.
- **method** – 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', 'c4'], metadata={'key': 'val'})
>>> app.inputs.update(new_img, action='overwrite')
```

## Models

`class clarifai.rest.client.Models` (*api*)

**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.

**Returns** None

**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')
```

## Model

**class** clarifai.rest.client.**Model** (*api, item=None, model\_id=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 overwriting 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')
```

**get\_concept\_ids** ()  
get concepts IDs associated with the model

**Parameters** **Void** –

**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

**Parameters** **void** –

**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** – input for the feedback

**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** – input for the feedback

**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

**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", "↩️"])
```

## 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

## Authentication

**class** clarifai.rest.client.**Auth**(*api*)  
Clarifai Authentication

This class is initialized as an attribute of the clarifai application object, accessed with app.auth

**get\_token()**  
get token string

**Returns** The token as a string

## Exceptions

**class** clarifai.rest.client.**ApiError**(*resource, params, method, response*)  
API Server error

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

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

## A

add\_concepts() (clarifai.rest.client.Inputs method), 13  
 add\_concepts() (clarifai.rest.client.Model method), 26  
 add\_term() (clarifai.rest.client.SearchQueryBuilder method), 31  
 ApiClientError (class in clarifai.rest.client), 32  
 ApiError (class in clarifai.rest.client), 32  
 Auth (class in clarifai.rest.client), 32

## B

bulk\_create() (clarifai.rest.client.Concepts method), 11  
 bulk\_create\_images() (clarifai.rest.client.Inputs method), 13  
 bulk\_delete\_concepts() (clarifai.rest.client.Inputs method), 13  
 bulk\_merge\_concepts() (clarifai.rest.client.Inputs method), 14  
 bulk\_update() (clarifai.rest.client.Concepts method), 11  
 bulk\_update() (clarifai.rest.client.Inputs method), 14

## C

check\_status() (clarifai.rest.client.Inputs method), 14  
 check\_upgrade() (clarifai.rest.client.ClarifaiApp method), 9  
 ClarifaiApp (class in clarifai.rest.client), 9  
 clear\_model\_cache() (clarifai.rest.client.Models method), 23  
 Concepts (class in clarifai.rest.client), 11  
 create() (clarifai.rest.client.Concepts method), 11  
 create() (clarifai.rest.client.Models method), 23  
 create\_image() (clarifai.rest.client.Inputs method), 15  
 create\_image\_from\_base64() (clarifai.rest.client.Inputs method), 15  
 create\_image\_from\_bytes() (clarifai.rest.client.Inputs method), 15  
 create\_image\_from\_filename() (clarifai.rest.client.Inputs method), 16  
 create\_image\_from\_url() (clarifai.rest.client.Inputs method), 16

## D

delete() (clarifai.rest.client.Inputs method), 17  
 delete() (clarifai.rest.client.Models method), 24  
 delete\_all() (clarifai.rest.client.Inputs method), 17  
 delete\_all() (clarifai.rest.client.Models method), 24  
 delete\_concepts() (clarifai.rest.client.Inputs method), 17  
 delete\_concepts() (clarifai.rest.client.Model method), 26  
 delete\_version() (clarifai.rest.client.Model method), 26  
 dict() (clarifai.rest.client.SearchQueryBuilder method), 32

## G

get() (clarifai.rest.client.Concepts method), 12  
 get() (clarifai.rest.client.Inputs method), 17  
 get() (clarifai.rest.client.Models method), 24  
 get\_all() (clarifai.rest.client.Concepts method), 12  
 get\_all() (clarifai.rest.client.Inputs method), 17  
 get\_all() (clarifai.rest.client.Models method), 24  
 get\_by\_page() (clarifai.rest.client.Concepts method), 12  
 get\_by\_page() (clarifai.rest.client.Inputs method), 18  
 get\_by\_page() (clarifai.rest.client.Models method), 25  
 get\_concept\_ids() (clarifai.rest.client.Model method), 26  
 get\_info() (clarifai.rest.client.Model method), 27  
 get\_inputs() (clarifai.rest.client.Model method), 27  
 get\_outputs() (clarifai.rest.client.Inputs method), 18  
 get\_token() (clarifai.rest.client.Auth method), 32  
 get\_version() (clarifai.rest.client.Model method), 27

## I

init\_model\_cache() (clarifai.rest.client.Models method), 25  
 Inputs (class in clarifai.rest.client), 13  
 InputSearchTerm (class in clarifai.rest.client), 30

## L

list\_versions() (clarifai.rest.client.Model method), 27

## M

merge\_concepts() (clarifai.rest.client.Inputs method), 18

`merge_concepts()` (clarifai.rest.client.Model method), 28  
`merge_metadata()` (clarifai.rest.client.Inputs method), 19  
`merge_outputs_concepts()` (clarifai.rest.client.Inputs method), 19

`Model` (class in clarifai.rest.client), 26  
`Models` (class in clarifai.rest.client), 23

## O

`OutputSearchTerm` (class in clarifai.rest.client), 31

## P

`predict()` (clarifai.rest.client.Model method), 28  
`predict_by_base64()` (clarifai.rest.client.Model method), 28  
`predict_by_bytes()` (clarifai.rest.client.Model method), 28  
`predict_by_filename()` (clarifai.rest.client.Model method), 28  
`predict_by_url()` (clarifai.rest.client.Model method), 29

## R

`remove_outputs_concepts()` (clarifai.rest.client.Inputs method), 19

## S

`search()` (clarifai.rest.client.Concepts method), 12  
`search()` (clarifai.rest.client.Inputs method), 19  
`search()` (clarifai.rest.client.Models method), 25  
`search_by_annotated_concepts()` (clarifai.rest.client.Inputs method), 19  
`search_by_geo()` (clarifai.rest.client.Inputs method), 20  
`search_by_image()` (clarifai.rest.client.Inputs method), 20  
`search_by_metadata()` (clarifai.rest.client.Inputs method), 21  
`search_by_original_url()` (clarifai.rest.client.Inputs method), 21  
`search_by_predicted_concepts()` (clarifai.rest.client.Inputs method), 22  
`SearchQueryBuilder` (class in clarifai.rest.client), 31  
`SearchTerm` (class in clarifai.rest.client), 30  
`send_concept_feedback()` (clarifai.rest.client.Model method), 29  
`send_region_feedback()` (clarifai.rest.client.Model method), 29  
`send_search_feedback()` (clarifai.rest.client.Inputs method), 22

## T

`tag_files()` (clarifai.rest.client.ClarifaiApp method), 10  
`tag_urls()` (clarifai.rest.client.ClarifaiApp method), 10  
`train()` (clarifai.rest.client.Model method), 29

## U

`update()` (clarifai.rest.client.Concepts method), 12

`update()` (clarifai.rest.client.Inputs method), 22  
`update()` (clarifai.rest.client.Model method), 29  
`UserError` (class in clarifai.rest.client), 32

## W

`wait_until_inputs_delete_finish()` (clarifai.rest.client.ClarifaiApp method), 10  
`wait_until_inputs_upload_finish()` (clarifai.rest.client.ClarifaiApp method), 10  
`wait_until_models_delete_finish()` (clarifai.rest.client.ClarifaiApp method), 11