contacthub Documentation

Release 0.0

Axant

Jun 06, 2018

Contents

This is the official Python SDK for ContactHub REST API. This SDK easily allows to access your data on ContactHub, making the authentication immediate and simplifying read/write operations.

For all information about ContactHub, check here.

Contents:

CHAPTER 1

Getting started

1.1 Installing the SDK

The ContactHub SDK can be installed from PyPi:

```
pip install contacthub
```

After installing, for importing the contacthub SDK just:

```
import contacthub
```

1.2 Performing simple operations on customers

1.2.1 Getting Customer's data

Retrieving entity's data can be easily achieved with simple operations.

First of all, you need to authenticate with credentials provided by *ContactHub*:

```
from contacthub import Workspace

workspace = Workspace(workspace_id='workspace_id', token='token')
```

After that you can get a *Node* object to perform all operations on customers and events:

```
node = workspace.get_node(node_id='node_id')
```

With a node, is immediate to get all customers data in a `list` of `Customer` objects:

```
customers = node.get_customers()

for customer in customers:
    print(customer.base.firstName)
```

Getting a single `Customer`:

```
my_customer = node.get_customer(id='id')

print('Welcome back %s' % my_customer.base.firstName)
```

or querying on customers by theirs own attributes:

```
fetched_customers = node.query(Customer).filter((Customer.base.firstName == 'Bruce') &
↪ (Customer.base.secondName == 'Wayne')).all()
```

1.2.2 Add a new Customer

Creating and posting a Customer is simple as getting. The method *add_customer* of the node take a dictionary containing the structure of your customer as parameter and returns a new Customer object:

```
customer_struct =    {
                    'base': {'contacts': {'email': 'myemail@email.com'}},
                    'extra': 'extra',
                    'extended': {'my_string':'my new extended property string'}
                    }
my_customer = c.add_customer(**customer_struct)
```

For creating the customer structure, you can also create a new Customer object and convert it to a dictionary for posting:

```
from contacthub.models import Customer

my_customer = Customer(node = node)
my_customer.base.contacts.email = 'myemail@email.com'
my_customer.extra = 'extra'
my_customer.extended.my_string = 'my new extended property string'
my_customer = node.add_customer(**my_customer.to_dict())
```

or posting it directly with the *post* method:

```
my_customer.post()
```

1.2.3 Relationship between Customers and Events

In this SDK entities are easily connected. For retrieving all events associated to a `Customer`, just:

```
my_customer = node.get_customer(id='id')
events = my_customer.get_events()
```

Note that relations are immutable objects. You can just consult events associated to a `Customer`, but you cannot add new ones or delete.

CHAPTER 2

Authentication

You can create a *Workspace* object that allows the control of the workspace's nodes. It require the workspace id and the access token provided by ContactHub:

```
my_workspace = Workspace(workspace_id='workspace_id', token='token')
```

If not specified, the SDK will use the default URL for the ConctactHub API - *https://api.contactlab.it/hub/v1* - but you can specify a different base URL for the API:

```
my_workspace = Workspace(workspace_id='workspace_id', token='token', base_url='base_
↪url')
```

Once obtained a workspace, you're able to access the various nodes linked to it with the *get_node* method:

```
node = workspace.get_node(node_id='node_id')
```

This method will return a *Node* object, that allows you to perform all operations on customers and events. A `Node` is a key object for getting, posting, putting, patching and deleting data on entities.

2.1 Authenticating via configuration file

You can specify the workspace ID, the access token and the base url (not mandatory. If ommited, the default base URL for ContactHub will be used) via INI file:

```
my_workspace = Workspace.from_INI_file('file.INI')
```

The file must follow this template:

```
workspace_id = workspace_id
token = token
base_url = base_url
```

2.2 Proxies

If you need to use a proxy, you can configure requests by setting the environment variables HTTP_PROXY and HTTPS_PROXY:

```
$ export HTTP_PROXY="http://10.10.1.10:3128"
$ export HTTPS_PROXY="http://10.10.1.10:1080"
```

To use HTTP Basic Auth with your proxy, use the *http://user:password@host/* syntax:

```
$ export HTTPS_PROXY="http://user:pass@10.10.1.10:3128/"
```

CHAPTER 3

Operations on Customers

After the *Authentication* you are ready to perform all operations on ContactHub's entities.

3.1 Create and add a new customer

Like every other entities in ContactHub, you can perform an operation via two methods:

1. Via the methods provided by the *Node* class
2. Performing the operation directly by your entity's object

3.1.1 1. Adding a new customer via the methods provided by the *Node* class

In this first case, a new customer can be added in ContactHub by the *Node* object. By default, the *add_customer* method takes as parameter a dictionary containing the structure of your new customer and return a new *Customer* object:

```
customer_structure = {
                     'externalId': '01',
                     'extra': 'extra',
                     'base': {
                             'timezone': 'Europe/Rome',
                             'firstName': 'Bruce',
                             'lastName': 'Wayne',
                             'contacts': {
                                         'email': 'email@email.com'
                                         }
                             }
                      }

my_customer = node.add_customer(**customer_structure)
```

To specify the structure of your new customer, you can also use the *Customer* class, creating a new *Customer* object and converting it to a dictionary:

```
from contacthub.models import Customer

post_customer = Customer(node = node)
post_customer.base.firstName = 'Bruce'
post_customer.base.lastName = 'Wayne'
post_customer.base.contacts.email = 'email@example.com'
post_customer.extra = 'extra'
post_customer.extended.my_string = 'my new extended property string'
post_customer.consents = {
    'marketing': {
        'automatic': {
            'email': {'status': True, 'limitation': False, 'objection': False}
        }
    }
}

new_customer = node.add_customer(**post_customer.to_dict())
```

When you declare a new *Customer*, by default its internal structure start with this template:

```
{'base':{
        'contacts': {}
        },
'extended': {},
'consents': {},
'tags': {
        'manual':[],
        'auto':[]
        }
}
```

You can directly access every simple attribute (strings, numbers) in a new customer created with the above structure.

It's possibile to re-define your own internal structure for a customer with the *default_attributes* parameter of the *Customer* constructor:

```
c = Customer(node=node, default_attributes={'base':{}})
```

In this case, you can directly set the *base* attribute, but you have to define beforehand all other objects in the internal structure.

Properties class

An important tool for this SDK it's the *Properties* class. It represents a default generic object and you should use it for simplify the declarations of entity's properties. In *Properties* object constructor you can declare every field you need for creating new properties. These fields can be strings, integer, datetime object, other *Properties* and lists of above types.

For example:

```
from contacthub.models import Properties

my_customer = Customer(node=node)
my_customer.base.contacts = Properties(email='bruce.wayne@darkknight.it', fax='fax',␣
↪otherContacts=[Properties(value='123', name='phone', type='MOBILE')])
```

(continues on next page)

(continued from previous page)

```
my_customer.base.address = Properties(city='city', province='province',␣
↪geo=Properties(lat=40, lon=100))

my_customer.post()
```

Extended properties

By default the extended properties are already defined in the *Customer* structure, so you can populate it with new integers, strings or *Properties* object for storing what you need. Extended properties follow a standardized schema defined in the ContactHub settings.

```
my_customer.extended.my_extended_int = 1
my_customer.extended.my_extended_string = 'string'
my_customer.extended.my_extended_object = Properties(key='value', k='v')
```

3.1.2 2. Posting a customer directly by its object

In the second case, after the creation of the *Customer* you can post it directly with the *post* method:

```
my_customer.post()
```

3.1.3 Force update

If the customer already exists in the node, you can force its update with the new structure specified. If the system notice a match between the new customer posted and an existent one in the node, with the flag *force_update* set to True, the customer will be updated with new data.:

```
my_customer = node.add_customer(**customer_structure, forceUpdate=True)
```

or alternatively:

```
my_customer.post(forceUpdate=True)
```

The match criteria between customers is a configurable option in the ContactHub settings.

For adding a new customer, you have to define its structure with all attributes you need. You must specify all required attribute, according to your ContactHub configuration. You can find the required attributes in your ContactHub dashboard.

N.B.: You must follow the ContatHub schema selected for your base properties. Check the ContactHub dashboard for further information.

For errors related to the addition of customers, see *Exception handling*.

3.2 Get all customers

To retrieve a list of customers in a node, just:

```
customers = node.get_customers()
```

This method return a list of *Customer* objects.

For example, for accessing the email of a customer:

```
print(my_customer.base.contacts.email)
```

or getting the manual tags associated to a customer in a list:

```
for tag in my_customer.tags.manual:
    print(tag)
```

In this way you can access every attribute of a single *Customer*.

Note that if you'll try to access for example the *base* attribute of a *Customer*, it will return an *Properties* object, that will contain all the base properties of the *Customer* object.

3.2.1 Paging the customers

When you retrieve a list of entities (e.g. *get_customers*) , a new *PaginatedList* object will be returned. The *PaginatedList* object allows you scrolling through the result pages from the API. By default you'll get the first 10 elements of total result, coming from the first page, but you can specify the maximum number of customers per page and the page to get.

For example, if you have 50 customers and you want to divide them in 5 per page, getting only the second page, use the *size* and the *page* parameters in this way:

```
customers = node.get_customers(size=5, page=2)
```

This call will return a *PaginatedList* of 5 customers, taken from the second subset (size 5) of 50 total customers.

Now you can navigate trough the result pages with two metods:

```
customers.next_page()

customer.previous_page()
```

By these two methods you can navigate through pages containing *Customers* object. The number of Customers for each page is determined by the *size* parameter of the *get_customer*, by default 10.

In a *PaginatedList* object you can find these attributes:

- *size*: the number of elements per each page
- *total_elements*: the number of total elements obtained
- *total_pages*: the number of total pages in wich are divided the elements
- *total_unfiltered_elements*: the element excluded from this set of elements
- *page_number*: the number of the current page. For increment it or decrement it, use the *next_page* and the *previous_page* methods.

Note that a *PaginatedList* is immutable: you can only read the elements from it and adding or removing elements to the list is not allowed.

3.2.2 Get a customer by their externalId

You can obtain a *PaginatedList* of *Customer* objects associated to an external ID by:

```
customers = node.get_customers(external_id="01")
```

If there's only one customer associated to the given external ID, this method will create a single *Customer* object instead of a *PaginatedList*

3.2.3 Get specific fields of customers

It's possible to filter the fields present in a *Customer*, specifying them in a list strings representing the attributes:

```
node.get_customers(fields=['base.firstName', 'base.lastName'])
```

None of the previous parameter passed to the 'get_customers' method is required and you can combine them for getting the list of customers that suits your needs.

3.3 Get a single customer

You can get a single customer by specifying its *id* or *externalId*, obtaining a new *Customer* object.

By id:

```
my_customer = node.get_customer(id='01')
```

or by the externalId:

```
my_customer = node.get_customer(external_id='02')
```

3.4 Query

3.4.1 Simple queries

ContactHub allows you to retrieve subsets of customers entry in a node, by querying on *Customer* entity.

To retrieve a list of Customers that satisfy your fetching criteria, just create a new *Query* object:

```
new_query = node.query(Customer)
```

Now you're ready to apply multiple filters on this *Query*, specifying new criteria as parameter of the .*filter'method of 'Query* class:

```
new_query = new_query.filter((Customer.base.firstName == 'Bruce') & (Customer.base.
↪lastName == 'Wayne'))
```

Each filter applied subsequently will put your new criteria in the *AND* condition, adding it to the criteria already present in the query:

```
new_query = new_query.filter((Customer.base.dob <= datetime(1994, 6, 10))
```

Once obtained a full filtered query, call the *.all()* method to apply the filters and get a *PaginatedList* of queried customers:

filtered_customers = new_query.all()

3.4.2 Available operations for creating a filter

Criteria	Operator
EQUAL	==
NE	!=
GT	>
GTE	>=
LT	<
LTE	<=
IN	function *in_* in contacthub.query module
NOT_IN	function *not_in_* in contacthub.query module
BETWEEN	function *between_* in contacthub.query module
IS_NULL	== None
IS_NOT_NULL	!= None

Equality operator

```
new_query = node.query(Customer).filter(Customer.base.firstName == 'Bruce')
```

Not equals

```
new_query = node.query(Customer).filter(Customer.base.firstName != 'Bruce')
```

Greater than

```
new_query = node.query(Customer).filter(Customer.base.dob > datetime(1994,6,10))
```

Greater than or equal

```
new_query = node.query(Customer).filter(Customer.base.dob >= datetime(1994,6,10))
```

Less than

```
new_query = node.query(Customer).filter(Customer.registeredAt < datetime(2010,6,10))
```

Less than or equal

```
new_query = node.query(Customer).filter(Customer.registeredAt <= datetime(2010,6,10))
```

In, Not in

You can verify the presence of a specific value in a customer attribute with the *in_* and *not_in_* methods of the *query* module:

```
from contacthub.models.query import in_

new_query = node.query(Customer).filter(in_('manual_tag', Customer.tags.manual))
```

```
from contacthub.models.query import not_in_

new_query = node.query(Customer).filter(not_in_('manual_tag', Customer.tags.manual))
```

Between

You can check if a customer date attribute is between two dates. These two dates can be *datetime* objects or normal string following the ISO8601 standard for dates.

```
from contacthub.models.query import between_

new_query = node.query(Customer).filter(between_(Customer.base.dob, datetime(1950,1,
↪1), datetime(1994,1,1)))
```

Is null

```
new_query = node.query(Customer).filter(Customer.base.firstName == None)
```

Is not null

```
new_query = node.query(Customer).filter(Customer.base.firstName != None)
```

3.4.3 Combine criteria

To combine the above criteria and create complex ones, you can use the *&* and | operators:

AND

```
customers = node.query(Customer).filter((Customer.base.firstName == 'Bruce') &␣
↪(Customer.base.lastName == 'Wayne')).all()
```

OR

```
customers = node.query(Customer).filter(((Customer.base.firstName == 'Bruce')) |␣
↪((Customer.base.firstName == 'Batman'))).all()
```

3.4.4 Combined query

It's possibile to combine simple queries to create a combined query. For this purpose, you can use the *&* operator to put two simple queries in the *AND* condition and the | operator for putting them in the *OR* condition:

```
q1 = node.query(Customer).filter(Customer.base.firstName == 'Bruce')
q2 = node.query(Customer).filter(Customer.base.lastName == 'Wayne')

and_query = q1 & q2

or_query = q1 | q2
```

For apply all filters created in the new combined query, just like the simple queries call the *.all()*:

> filtered_customers = and_query.all()

3.5 Update a customer

Customers can be updated with new data. The update can be carried on an entire customer or only on a few attributes.

3.5.1 Full update - Put

The full update on customer - PUT method - totally replace old customer attributes with new ones. As all operations on this SDK, you can perform the full update in two ways: by the the methods in the *Node* class or directly by the *Customer* object.

Note that if you perform the full update operation by the *update_customer* method of the node, you have to pass all attributes previously set on the customer, otherwise an APIError will occur (see *Exception handling*). These attributes can be easily retrieved via the *to_dict* method.

Set the *full_update* flag to *True* for a full update, eg:

```
my_customer = node.get_customer(id='id')
my_customer.base.contacts.email = 'anotheremail@example.com'

updated_customer = node.update_customer(**my_customer.to_dict(), full_update=True)
```

To directly execute a full update on a customer by the *Customer* object:

```
my_customer = node.get_customer(id='customer_id')
my_customer.base.contacts.email = 'anotheremail@example.com'

my_customer.put()
```

There are no difference between these two ways of working. By default the parameter *full_update* is set to False, so without specifying it you'll perform a partial update (see the next section **Partial update - Patch**).

3.5.2 Partial update - Patch

The partial update - PATCH method - applies partial modifications to a customer.

Since all list attributes don't allow normal list operation (*append*, *reverse*, *pop*, *insert*, *remove*, *__setitem__*, *__delitem__*, *__setslice__*), for adding an element in an existing list attribute of a customer, you can use the += operator:

```
customer.base.subscriptions += [Properties(id='id', name='name', type='type',␣
↪kind=Cutomer.SUBSCRPTION_KINDS.SERVICE)]
```

Once the customer is modified, you can get the changes occurred on its attributes by the *get_mutation_tracker* method, that returns a new dictionary:

```
my_customer = node.get_customer(id='customer_id')
my_customer.base.contacts.email = 'anotheremail@example.com'

updated_customer = node.update_customer(**my_customer.get_mutation_tracker())
```

You can also pass to the *update_customer* method a dictionary representing the mutations you want to apply on customer attributes and the id of the customer for applying it:

```
mutations = {'base':{'contacts':{'email':'anotheremail@example.com'}}}

updated_customer = node.update_customer(id='customer_id',**mutations)
```

To partially update a customer by the *Customer* object, just:

```
my_customer.base.contacts.email = 'anotheremail@example.com'

my_customer.patch()
```

3.6 Delete a customer

Via the node method, passing the id of a customer:

```
node.delete_customer(id='customer_id')
```

or passing the dictionary form of the customer:

```
node.delete_customer(**my_customer.to_dict())
```

Via *Customer* object:

```
my_customer.delete()
```

3.7 Tags

Tags are particular string values stored in two arrays: *auto* (autogenerated from elaborations) and *manual* (manually inserted). To get the tags associated to a customer, just access the *tags* attribute of a *Customer* object:

```
for auto in my_customer.tags.auto:
    print(auto)

for manual in my_customer.tags.manual:
    print(manual)
```

The *Node* class provides two methods for inserting and removing *manual* tags:

```
node.add_tag('manual_tag')
```

When removing a manual tag, if it doesn't exists in the customer tags a ValueError will be thrown:

```
try:
    node.remove_tag('manual_tag')
except ValueError as e:
        #actions
```

3.8 Additional entities

ContactHub provides three endpoints to reach some particular and relevant attributes of a customer. These endpoint simplify the add, the delete, the update and the get operations of *educations* , *likes*, *jobs* and *subscriptions* base attributes. For this purpose, this SDK provides three additional classes for managing these attributes:

- *Education*
- *Job*
- *Like*
- *Subscription*

You can operate on these classes alike other entities (*Customer* and *Event*): via the methods of the *Node* class or directly by the classes. These entities are identified by an internal ID and have their own attributes.

3.9 Education

3.9.1 Get

You can get an education associated to a customer by the customer ID and an education ID previously assigned to the education:

```
customer_education = node.get_education(customer_id='c_id', education_id='education_id
↪')
```

This method creates an *Education* object. You can find the same object in the list of the educations for a customer, accessing the *base.educations* attribute of a *Customer* object.

3.9.2 Add

Add via the node method, creating a new *Education* object:

```
new_educ = node.add_education(customer_id='123', id='01', schoolType=Education.SCHOOL_
↪TYPES.COLLEGE,
schoolName='schoolName',schoolConcentration='schoolConcentration', isCurrent=False,␣
↪startYear='1994', endYear='2000')
```

or directly by the object:

```
from contacthub.model import Education

new_educ = Education(customer=my_customer, id='01', schoolType=Education.SCHOOL_TYPES.
↪COLLEGE, schoolName='schoolName',
schoolConcentration='schoolConcentration', isCurrent=False, startYear='1994', endYear=
↪'2000')
```

(continues on next page)

(continued from previous page)

```
new_educ.post()
```

3.9.3 Remove

Remove via the node method:

```
node.remove_education(customer_id='c_id', education_id='education_id')
```

or directly by the object:

```
education.delete()
```

3.9.4 Update

After some changes on a *Education*:

```
my_education = node.get_education(customer_id='c_id', education_id='education_id')
my_education.schoolConcentration = 'updated'
```

you can update it via the node method:

```
node.update_education(customer_id='c_id', **my_education.to_dict())
```

or directly by the object:

```
my_education.put()
```

3.10 Job

3.10.1 Get

You can get a job associated to a customer by the customer ID and a job ID:

```
customer_job = node.get_job(customer_id='c_id', job_id='job_id')
```

This method creates a *Job* object.

3.10.2 Add

Add via the node method, creating a new *Job* object:

```
new_job = node.add_job(customer_id='123', id='01', jobTitle='jobTitle', companyName=
↪'companyName',
companyIndustry='companyIndustry', isCurrent=True, startDate='1994-10-06', endDate=
↪'1994-10-06')
```

or directly by the object:

```
new_job = Job(customer=my_customer, id='01', jobTitle='jobTitle', companyName=
↪'companyName', companyIndustry='companyIndustry',
isCurrent=True, startDate='1994-10-06', endDate='1994-10-06')

new_job.post()
```

3.10.3 Remove

Remove via the node method:

```
node.remove_job(customer_id='c_id', job_id='job_id')
```

or directly by the object:

```
job.delete()
```

3.10.4 Update

After some changes on a *Job*:

```
my_job = node.get_job(customer_id='c_id', job_id='job_id')
my_job.jobTitle = 'updated'
```

you can update it via the node method:

```
node.update_job(customer_id='c_id', **my_job.to_dict())
```

or directly by the object:

```
my_job.put()
```

3.11 Like

3.11.1 Get

You can get a like associated to a customer by the customer ID and a like ID:

```
my_like = node.get_like(customer_id='c_id', like_id='like_id')
```

This method creates a *Like* object.

3.11.2 Add

Add via the node method, creating a new *Like* object:

```
new_like= node.add_like(customer_id='123', id='01', name='name', category='category',
createdTime=datetime.now())
```

or directly by the object:

```
new_like = Like(customer=my_customer, id='01', name='name', category='category',␣
↪createdTime=datetime.now())

new_like.post()
```

3.11.3 Remove

Remove via the node method:

```
node.remove_like(customer_id='c_id', like_id='like_id')
```

or directly by the object:

```
like.delete()
```

3.11.4 Update

After some changes on a *Like*:

```
my_like = node.get_like(customer_id='c_id', like_id='like_id')
my_like.name = 'updated'
```

you can update it via the node method:

```
node.update_like(customer_id='c_id', **my_like.to_dict())
```

or directly by the object:

```
my_like.put()
```

3.12 Subscription

3.12.1 Get

You can get a subscription associated to a customer by the customer ID and a subscription ID previously assigned to the subscription:

```
customer_sub = node.get_subscription(customer_id='c_id', subscription_id=
↪'subscription_id')
```

3.12.2 Add

Add via the node method, creating a new *Subscription* object:

```
new_sub = node.add_subscription(customer_id='01', id='02', name='name',␣
↪kind=Subscription.KINDS.SERVICE,
subscriberId='id', subscribed=True, preferences=[{'key':'value'}])
```

or directly by the object:

```
new_sub = Subscription(customer=my_customer, id='02', name='name', kind=Subscription.
↪KINDS.SERVICE,
subscriberId='id', subscribed=True, preferences=[{'key':'value'}])

new_sub.post()
```

3.12.3 Remove

Remove via the node method:

```
node.remove_subscription(customer_id='c_id', subscription_id='subscription_id')
```

or directly by the object:

```
subscription.delete()
```

3.12.4 Update

After some changes on a *Subscription*:

```
my_sub = node.get_subscription(customer_id='c_id', subscription_id='subscription_id')
my_sub.name = 'updated'
```

you can update it via the node method:

```
node.update_subscription(customer_id='c_id', **my_sub.to_dict())
```

or directly by the object:

```
my_sub.put()
```

CHAPTER 4

Operations on Events

The second important entity of ContactHub are events, particular actions performed by customers in particular contexts.

The attribute *type* of an *Event* defines the schema that you must follow for his attributes. His schema must be specified in the *properties* attribute of the *Event*.

Eg: If you create a new event whose type is *serviceSubscribed*, you must specify these attributes in the *properties* attribute of the Event:

Name	Type
subscriberId	string
serviceId	string
serviceName	string
serviceType	string
startDate	string
endDate	string
extraProperties	object

Active event types and their related schemas can be found on ContactHub Settings

Events are also associated to a context of use. The available contexts for an event are:

- CONTACT_CENTER
- WEB
- MOBILE
- ECOMMERCE
- RETAIL
- IOT
- SOCIAL
- DIGITAL_CAMPAIGN

- OTHER

When you create an *Event*, the attributes *type*, *context* and *properties* are required.

For each type of *context*, you can also provide some additional properties in the optional *contextInfo* object. The schema for this object varies depending on the specified *context*. All the schemas can be found at this link.

4.1 Add a new event

To create a new event, you have to define its schema (according to the specified type) in *Event* class constructor:

```
event = Event(
    node=node,
    customerId=my_customer.id,
    type=Event.TYPES.SERVICE_SUBSCRIBED,
    context=Event.CONTEXTS.WEB,
    contextInfo=Properties(
        client=Properties(
            ip="127.0.0.1"
        )
    ),
    properties=Properties(
        subscriberId='s_id', serviceId='service_id', serviceName='serviceName',␣
↪startDate=datetime.now(),
        extraProperties=Properties(extra='extra')
    )
)
```

The attribute ‘customerId‘ is required for specifying the customer who will be associated with the event.

After the creation, you can add the event via the node method:

```
posted = node.add_event(**event.to_dict())
```

or directly by the object:

```
event.post()
```

For some events, you have to specify the *mode* attribute. Its value must be:

- ACTIVE: if the customer made the event
- PASSIVE: if the customer receive the event

4.2 Sessions

ContactHub allows saving anonymous events of which it is expected that the customer will be identified in the future. For this purpose, you can save an event using a Session ID, an unique identifier that is assigned to the anonymous customer, without specify the customer ID.

To save an event with a Session ID, use the *bringBackProperties* attribute.

You can generate a new session ID conforming to the UUID standard V4 by the *create_session_id* method of the *Node*:

```
session_id = node.create_session_id()

event = Event(node=node, bringBackProperties=Properties(type='SESSION_ID',␣
↪value=session_id),
type=Event.TYPES.SERVICE_SUBSCRIBED, context=Event.CONTEXTS.WEB,
properties=Properties(subscriberId = 's_id', serviceId='service_id', serviceName=
↪'serviceName', startDate=datetime.now(),
extraProperties=Properties(extra='extra')))
```

If the anonymous customer will perform the authentication, you can add a new *Customer* on ContactHub, obtaining a new customer ID. To reconcile the events stored with the session ID previously associated to the anonymous Customer:

```
node.add_customer_session(customer_id='c_id', session_id=session_id)
```

In this way, every event stored with the same session ID specified as parameter in the *add_customer_session* method, will be associated to the customer specified.

4.3 ExternalId

In the same way of the Session ID, you can add a new event specifying the external ID of a customer:

```
from contacthub.models import Event

event = Event(node=node, bringBackProperties=Properties(type='externalId', value='e_id
↪'),
type=Event.TYPES.SERVICE_SUBSCRIBED, context=Event.CONTEXTS.WEB,
properties=Properties(subscriberId = 's_id', serviceId='service_id', serviceName=
↪'serviceName', startDate=datetime.now(),
extraProperties=Properties(extra='extra')))
```

4.4 Get all events

To get all events associated to a customer, use *Node* method:

```
events = node.get_events(customer_id='c_id')
```

You can filter events specifying the following parameters in *get_events* method:

- event_type
- context
- event_mode
- date_from
- date_to
- page
- size

```
events = node.get_events(customer_id='c_id', event_type=Event.TYPES.SERVICE_
↪SUBSCRIBED, context=Event.CONTEXTS.WEB)
```

This method will return a *PaginatedList* (see *Paging the customers*).

A shortcut for customer events is available as a property in a *Customer* object:

```
for event in my_customer.get_events():
    print (event.type)
```

In this last case, the property will return an immutable list of *Event*: you can only read the events associated to a customer from it and adding events to the list is not allowed.

4.5 Get a single event

Retrieve a single event by its ID, obtaining a new *Event* object:

```
customer_event = event.get_event(id='event_id')
```

CHAPTER 5

Exception handling

When ContactHub API's returns a different status codes than 2xx Success, an *ApiError* Exception will be thrown. A sample of an *ApiError* message:

```
contacthub.errors.api_error.APIError: Status code: 409. Message: Conflict with␣
↪exiting customer 8b321dce-53c4-4029-8388-1938efa2090c. Errors: []. Data: data}.␣
↪Logref: logref_n
```

CHAPTER 6

API Reference

6.1 Authentication

6.1.1 Node

class `contacthub.node.`**`Node`**(*workspace*, *node_id*)

Bases: `object`

Node class for accessing data on a Contacthub node.

`add_customer`(*force_update=False*, ***attributes*)

Add a new customer in contacthub. If the customer already exist and force update is true, this method will update the entire customer with new data

Parameters

- **`attributes`** – the attributes for inserting the customer in the node
- **`force_update`** – a flag for update an already present customer

Returns the customer added or updated

`add_customer_session`(*customer_id*, *session_id*)

Add a new session id for a customer.

Parameters

- **`customer_id`** – the customer ID for adding the session id
- **`session_id`** – a session ID for create a new session

Returns the session id of the new session inserted

`add_education`(*customer_id*, ***attributes*)

Insert a new Education for the given Customer

Parameters

- **`customer_id`** – the id of the customer for adding the Education

- **attributes** – the attributes representing the new Education to add

Returns a Education object representing the added Education

add_event(***attributes*)

Add an event in this node. For adding it and associate with a known customer, specify the customer id in the attributes of the Event. For associate it to an external Id or a session id of a customer, specify in the bringBackProperties object like: {'type':'EXTERNAL_ID', 'value':'value', 'nodeId':'nodeId' }

Parameters **attributes** – the attributes of the event to add in the node

Returns a new Event object representing the event added in this node

add_job(*customer_id*, ***attributes*)

Insert a new Job for the given Customer

Parameters

- **customer_id** – the id of the customer for adding the job
- **attributes** – the attributes representing the new job to add

Returns a Job object representing the added Job

add_like(*customer_id*, ***attributes*)

Insert a new Like for the given Customer

Parameters

- **customer_id** – the id of the customer for adding the Like
- **attributes** – the attributes representing the new Like to add

Returns a Like object representing the added Like

add_subscription(*customer_id*, ***attributes*)

Insert a new Subscription for the given Customer

Parameters

- **customer_id** – the id of the customer for adding the Subscription
- **attributes** – the attributes representing the new Subscription to add

Returns a Subscription object representing the added Subscription

add_tag(*customer_id*, *tag*)

Add a new tag in the list of customer's tags

Parameters

- **customer_id** – the id customer in which adding the tag
- **tag** – a string, int, representing the tag to add

static create_session_id()

Create a new random session id conformed to the UUID standard

Returns a new session id conformed to the UUID standard

delete_customer(*id*, ***attributes*)

Delete the specified Customer from contacthub. For deleting an existing customer object, you should:

```
node.delete_customer(**c.to_dict())
```

Parameters

- **`id`** – a the id of the customer to delete
- **`attributes`** – the attributes of the customer to delete

Returns an object representing the deleted customer

`get_customer`(*id=None*, *external_id=None*)

Retrieve a customer from the associated node by its id or external ID. Only one parameter can be specified for getting a customer.

Parameters

- **`id`** – the id of the customer to retrieve
- **`external_id`** – the external id of the customer to retrieve

Returns a Customer object representing the fetched customer

`get_customer_education`(*customer_id*, *education_id*)

Get an education associated to a customer by its ID

Parameters

- **`education_id`** – the unique id of the education to get in a customer
- **`customer_id`** – the id of the customer for getting the education

Returns a new Education object containing the attributes associated to the education

`get_customer_job`(*customer_id*, *job_id*)

Get a job associated to a customer by its ID

Parameters

- **`job_id`** – the unique id of the job to get in a customer
- **`customer_id`** – the id of the customer for getting the job

Returns a new Job object containing the attributes associated to the job

`get_customer_like`(*customer_id*, *like_id*)

Get a like associated to a customer by its ID

Parameters

- **`like_id`** – the unique id of the like to get in a customer
- **`customer_id`** – the id of the customer for getting the like

Returns a new Like object containing the attributes associated to the like

`get_customer_subscription`(*customer_id*, *subscription_id*)

Get an subscription associated to a customer by its ID

Parameters

- **`subscription_id`** – the unique id of the subscription to get in a customer
- **`customer_id`** – the id of the customer for getting the subscription

Returns a new Subscription object containing the attributes associated to the subscription

`get_customers`(*external_id=None*, *page=None*, *size=None*, *fields=None*)

Get all the customers in this node

Parameters

- **`external_id`** – the external id of the customer to retrieve

- **`size`** – the size of the pages containing customers
- **`page`** – the number of the page for retrieve customer data
- **`fields`** – : a list of strings representing the properties to include in the response

Returns A list containing Customer object of a node

`get_event`(*id*)
Get a single event by its own id

Parameters **`id`** – the id of the event to get

Returns a new Event object representing the fetched event

`get_events`(*customer_id*, *event_type=None*, *context=None*, *event_mode=None*, *date_from=None*, *date_to=None*, *page=None*, *size=None*)
Get all events associated to a customer.

Parameters

- **`customer_id`** – The id of the customer owner of the event
- **`event_type`** – the type of the event present in Event.TYPES
- **`context`** – the context of the event present in Event.CONTEXT
- **`event_mode`** – the mode of event. ACTIVE if the customer made the event, PASSIVE if the customer recive the event
- **`date_from`** – From string or datetime for search of event
- **`date_to`** – From string or datetime for search of event
- **`size`** – the size of the pages containing events
- **`page`** – the number of the page for retrieve event data

Returns a list containing the fetched events associated to the given customer id

`query`(*entity*)
Create a QueryBuilder object for a given entity, that allows to filter the entity's data

Parameters **`entity`** – A class of model on which to run the query

Returns A QueryBuilder object for the specified entity

`remove_education`(*customer_id*, *education_id*)
Remove a the given Education for the given Customer

Parameters

- **`customer_id`** – the id of the customer associated to the Education to remove
- **`education_id`** – the id of the Education to remove

`remove_job`(*customer_id*, *job_id*)
Remove a the given Job for the given Customer

Parameters

- **`customer_id`** – the id of the customer associated to the job to remove
- **`job_id`** – the id of the job to remove

`remove_like`(*customer_id*, *like_id*)
Remove a the given Like for the given Customer

Parameters

- **`customer_id`** – the id of the customer associated to the Like to remove
- **`like_id`** – the id of the Like to remove

`remove_subscription`(*customer_id*, *subscription_id*)
Remove a the given Subscription for the given Customer

Parameters

- **`customer_id`** – the id of the customer associated to the Subscription to remove
- **`subscription_id`** – the id of the Subscription to remove

`remove_tag`(*customer_id*, *tag*)
Remove (if exists) a tag in the list of customer's tag

Parameters

- **`customer_id`** – the id customer in which adding the tag
- **`tag`** – a string, int, representing the tag to add

`update_customer`(*id*, *full_update=False*, ***attributes*)
Update a customer in contacthub with new data. If full_update is true, this method will update the full customer (PUT) and not only the changed data (PATCH)

Parameters

- **`id`** – the customer ID for updating the customer with new attributes
- **`full_update`** – a flag for execute a full update to the customer
- **`attributes`** – the attributes to patch or put in the customer

Returns the customer updated

`update_education`(*customer_id*, *id*, ***attributes*)
Update the given Education of the given customer with new specified attributes

Parameters

- **`customer_id`** – the id of the customer associated to the Education to update
- **`id`** – the id of the Education to update
- **`attributes`** – the attributes for update the Education

Returns a Education object representing the updated Education

`update_job`(*customer_id*, *id*, ***attributes*)
Update the given job of the given customer with new specified attributes

Parameters

- **`customer_id`** – the id of the customer associated to the job to update
- **`id`** – the id of the job to update
- **`attributes`** – the attributes for update the job

Returns a Job object representing the updated Job

`update_like`(*customer_id*, *id*, ***attributes*)
Update the given Like of the given customer with new specified attributes

Parameters

- **`customer_id`** – the id of the customer associated to the Like to update

- **id** – the id of the Like to update
- **attributes** – the attributes for update the Like

Returns a Like object representing the updated Like

update_subscription(*customer_id*, *id*, ***attributes*)
Update the given Subscription of the given customer with new specified attributes

Parameters

- **customer_id** – the id of the customer associated to the Subscription to update
- **id** – the id of the Subscription to update
- **attributes** – the attributes for update the Subscription

Returns a Subscription object representing the updated Subscription

6.1.2 Workspace

class contacthub.workspace.**Workspace**(*workspace_id*, *token*, *base_url='https://api.contactlab.it/hub/v1/workspaces'*)
Bases: object

Workspace class for authenticating on the specified base url APIs. This class is the first step for accessing the Contacthub APIs, the higher level.

classmethod from_ini_file(*file_path*)
Create a new Workspace object, taking the Workspace id, the token and the base URL from a INI configuration file

Parameters **file_path** – The path of the INI file for the parameters

Returns a new Workspace object

get_node(*node_id*)
Retrieve the node associated at the specified node id

Parameters **node_id** – The ID of the node to retrieve

Returns a Node object with the Workspace object specified

6.2 Customer

class contacthub.models.customer.**Customer**(*node*, *default_attributes=None*, ***attributes*)
Bases: object

Customer entity definition

class MOBILE_DEVICE_TYPES

APN = 'APN'

GCM = 'GCM'

WP = 'WP'

class NOTIFICATION_SERVICES

ADM = 'ADM'

```
    APN = 'APN'

    GCM = 'GCM'

    SNS = 'SNS'

    WNS = 'WNS'

class OTHER_CONTACT_TYPES

    EMAIL = 'EMAIL'

    FAX = 'FAX'

    MOBILE = 'MOBILE'

    OTHER = 'OTHER'

    PHONE = 'PHONE'

class TIMEZONES

    AFRICA_ABIDJAN = 'Africa/Abidjan'

    AFRICA_ACCRA = 'Africa/Accra'

    AFRICA_ALGIERS = 'Africa/Algiers'

    AFRICA_BISSAU = 'Africa/Bissau'

    AFRICA_CAIRO = 'Africa/Cairo'

    AFRICA_CASABLANCA = 'Africa/Casablanca'

    AFRICA_CEUTA = 'Africa/Ceuta'

    AFRICA_EL_AAIUN = 'Africa/El_Aaiun'

    AFRICA_JOHANNESBURG = 'Africa/Johannesburg'

    AFRICA_KHARTOUM = 'Africa/Khartoum'

    AFRICA_LAGOS = 'Africa/Lagos'

    AFRICA_MAPUTO = 'Africa/Maputo'

    AFRICA_MONROVIA = 'Africa/Monrovia'

    AFRICA_NAIROBI = 'Africa/Nairobi'

    AFRICA_NDJAMENA = 'Africa/Ndjamena'

    AFRICA_TRIPOLI = 'Africa/Tripoli'

    AFRICA_TUNIS = 'Africa/Tunis'

    AFRICA_WINDHOEK = 'Africa/Windhoek'

    AMERICA_ADAK = 'America/Adak'

    AMERICA_ANCHORAGE = 'America/Anchorage'

    AMERICA_ARAGUAINA = 'America/Araguaina'

    AMERICA_ARGENTINA_BUENOS_AIRES = 'America/Argentina/Buenos_Aires'

    AMERICA_ARGENTINA_CATAMARCA = 'America/Argentina/Catamarca'

    AMERICA_ARGENTINA_CORDOBA = 'America/Argentina/Cordoba'
```

```
AMERICA_ARGENTINA_JUJUY = 'America/Argentina/Jujuy'
AMERICA_ARGENTINA_LA_RIOJA = 'America/Argentina/La_Rioja'
AMERICA_ARGENTINA_MENDOZA = 'America/Argentina/Mendoza'
AMERICA_ARGENTINA_RIO_GALLEGOS = 'America/Argentina/Rio_Gallegos'
AMERICA_ARGENTINA_SALTA = 'America/Argentina/Salta'
AMERICA_ARGENTINA_SAN_JUAN = 'America/Argentina/San_Juan'
AMERICA_ARGENTINA_SAN_LUIS = 'America/Argentina/San_Luis'
AMERICA_ARGENTINA_TUCUMAN = 'America/Argentina/Tucuman'
AMERICA_ARGENTINA_USHUAIA = 'America/Argentina/Ushuaia'
AMERICA_ASUNCION = 'America/Asuncion'
AMERICA_ATIKOKAN = 'America/Atikokan'
AMERICA_BAHIA = 'America/Bahia'
AMERICA_BAHIA_BANDERAS = 'America/Bahia_Banderas'
AMERICA_BARBADOS = 'America/Barbados'
AMERICA_BELEM = 'America/Belem'
AMERICA_BELIZE = 'America/Belize'
AMERICA_BLANC_SABLON = 'America/Blanc-Sablon'
AMERICA_BOA_VISTA = 'America/Boa_Vista'
AMERICA_BOGOTA = 'America/Bogota'
AMERICA_BOISE = 'America/Boise'
AMERICA_CAMBRIDGE_BAY = 'America/Cambridge_Bay'
AMERICA_CAMPO_GRANDE = 'America/Campo_Grande'
AMERICA_CANCUN = 'America/Cancun'
AMERICA_CARACAS = 'America/Caracas'
AMERICA_CAYENNE = 'America/Cayenne'
AMERICA_CHICAGO = 'America/Chicago'
AMERICA_CHIHUAHUA = 'America/Chihuahua'
AMERICA_COSTA_RICA = 'America/Costa_Rica'
AMERICA_CRESTON = 'America/Creston'
AMERICA_CUIABA = 'America/Cuiaba'
AMERICA_CURACAO = 'America/Curacao'
AMERICA_DANMARKSHAVN = 'America/Danmarkshavn'
AMERICA_DAWSON = 'America/Dawson'
AMERICA_DAWSON_CREEK = 'America/Dawson_Creek'
AMERICA_DENVER = 'America/Denver'
AMERICA_DETROIT = 'America/Detroit'
```

```
AMERICA_EDMONTON = 'America/Edmonton'

AMERICA_EIRUNEPE = 'America/Eirunepe'

AMERICA_EL_SALVADOR = 'America/El_Salvador'

AMERICA_FORTALEZA = 'America/Fortaleza'

AMERICA_FORT_NELSON = 'America/Fort_Nelson'

AMERICA_GLACE_BAY = 'America/Glace_Bay'

AMERICA_GODTHAB = 'America/Godthab'

AMERICA_GOOSE_BAY = 'America/Goose_Bay'

AMERICA_GRAND_TURK = 'America/Grand_Turk'

AMERICA_GUATEMALA = 'America/Guatemala'

AMERICA_GUAYAQUIL = 'America/Guayaquil'

AMERICA_GUYANA = 'America/Guyana'

AMERICA_HALIFAX = 'America/Halifax'

AMERICA_HAVANA = 'America/Havana'

AMERICA_HERMOSILLO = 'America/Hermosillo'

AMERICA_INDIANA_INDIANAPOLIS = 'America/Indiana/Indianapolis'

AMERICA_INDIANA_KNOX = 'America/Indiana/Knox'

AMERICA_INDIANA_MARENGO = 'America/Indiana/Marengo'

AMERICA_INDIANA_PETERSBURG = 'America/Indiana/Petersburg'

AMERICA_INDIANA_TELL_CITY = 'America/Indiana/Tell_City'

AMERICA_INDIANA_VEVAY = 'America/Indiana/Vevay'

AMERICA_INDIANA_VINCENNES = 'America/Indiana/Vincennes'

AMERICA_INDIANA_WINAMAC = 'America/Indiana/Winamac'

AMERICA_INUVIK = 'America/Inuvik'

AMERICA_IQALUIT = 'America/Iqaluit'

AMERICA_JAMAICA = 'America/Jamaica'

AMERICA_JUNEAU = 'America/Juneau'

AMERICA_KENTUCKY_LOUISVILLE = 'America/Kentucky/Louisville'

AMERICA_KENTUCKY_MONTICELLO = 'America/Kentucky/Monticello'

AMERICA_LA_PAZ = 'America/La_Paz'

AMERICA_LIMA = 'America/Lima'

AMERICA_LOS_ANGELES = 'America/Los_Angeles'

AMERICA_MACEIO = 'America/Maceio'

AMERICA_MANAGUA = 'America/Managua'

AMERICA_MANAUS = 'America/Manaus'

AMERICA_MARTINIQUE = 'America/Martinique'
```

```
AMERICA_MATAMOROS = 'America/Matamoros'
AMERICA_MAZATLAN = 'America/Mazatlan'
AMERICA_MENOMINEE = 'America/Menominee'
AMERICA_MERIDA = 'America/Merida'
AMERICA_METLAKATLA = 'America/Metlakatla'
AMERICA_MEXICO_CITY = 'America/Mexico_City'
AMERICA_MIQUELON = 'America/Miquelon'
AMERICA_MONCTON = 'America/Moncton'
AMERICA_MONTERREY = 'America/Monterrey'
AMERICA_MONTEVIDEO = 'America/Montevideo'
AMERICA_NASSAU = 'America/Nassau'
AMERICA_NEW_YORK = 'America/New_York'
AMERICA_NIPIGON = 'America/Nipigon'
AMERICA_NOME = 'America/Nome'
AMERICA_NORONHA = 'America/Noronha'
AMERICA_NORTH_DAKOTA_BEULAH = 'America/North_Dakota/Beulah'
AMERICA_NORTH_DAKOTA_CENTER = 'America/North_Dakota/Center'
AMERICA_NORTH_DAKOTA_NEW_SALEM = 'America/North_Dakota/New_Salem'
AMERICA_OJINAGA = 'America/Ojinaga'
AMERICA_PANAMA = 'America/Panama'
AMERICA_PANGNIRTUNG = 'America/Pangnirtung'
AMERICA_PARAMARIBO = 'America/Paramaribo'
AMERICA_PHOENIX = 'America/Phoenix'
AMERICA_PORTO_VELHO = 'America/Porto_Velho'
AMERICA_PORT_AU_PRINCE = 'America/Port-au-Prince'
AMERICA_PORT_OF_SPAIN = 'America/Port_of_Spain'
AMERICA_PUERTO_RICO = 'America/Puerto_Rico'
AMERICA_RAINY_RIVER = 'America/Rainy_River'
AMERICA_RANKIN_INLET = 'America/Rankin_Inlet'
AMERICA_RECIFE = 'America/Recife'
AMERICA_REGINA = 'America/Regina'
AMERICA_RESOLUTE = 'America/Resolute'
AMERICA_RIO_BRANCO = 'America/Rio_Branco'
AMERICA_SANTAREM = 'America/Santarem'
AMERICA_SANTIAGO = 'America/Santiago'
AMERICA_SANTO_DOMINGO = 'America/Santo_Domingo'
```

```
AMERICA_SAO_PAULO = 'America/Sao_Paulo'

AMERICA_SCORESBYSUND = 'America/Scoresbysund'

AMERICA_SITKA = 'America/Sitka'

AMERICA_ST_JOHNS = 'America/St_Johns'

AMERICA_SWIFT_CURRENT = 'America/Swift_Current'

AMERICA_TEGUCIGALPA = 'America/Tegucigalpa'

AMERICA_THULE = 'America/Thule'

AMERICA_THUNDER_BAY = 'America/Thunder_Bay'

AMERICA_TIJUANA = 'America/Tijuana'

AMERICA_TORONTO = 'America/Toronto'

AMERICA_VANCOUVER = 'America/Vancouver'

AMERICA_WHITEHORSE = 'America/Whitehorse'

AMERICA_WINNIPEG = 'America/Winnipeg'

AMERICA_YAKUTAT = 'America/Yakutat'

AMERICA_YELLOWKNIFE = 'America/Yellowknife'

ANTARCTICA_CASEY = 'Antarctica/Casey'

ANTARCTICA_DAVIS = 'Antarctica/Davis'

ANTARCTICA_DUMONTDURVILLE = 'Antarctica/DumontDUrville'

ANTARCTICA_MACQUARIE = 'Antarctica/Macquarie'

ANTARCTICA_MAWSON = 'Antarctica/Mawson'

ANTARCTICA_PALMER = 'Antarctica/Palmer'

ANTARCTICA_ROTHERA = 'Antarctica/Rothera'

ANTARCTICA_SYOWA = 'Antarctica/Syowa'

ANTARCTICA_TROLL = 'Antarctica/Troll'

ANTARCTICA_VOSTOK = 'Antarctica/Vostok'

ASIA_ALMATY = 'Asia/Almaty'

ASIA_AMMAN = 'Asia/Amman'

ASIA_ANADYR = 'Asia/Anadyr'

ASIA_AQTAU = 'Asia/Aqtau'

ASIA_AQTOBE = 'Asia/Aqtobe'

ASIA_ASHGABAT = 'Asia/Ashgabat'

ASIA_BAGHDAD = 'Asia/Baghdad'

ASIA_BAKU = 'Asia/Baku'

ASIA_BANGKOK = 'Asia/Bangkok'

ASIA_BARNAUL = 'Asia/Barnaul'

ASIA_BEIRUT = 'Asia/Beirut'
```

ASIA_BISHKEK = 'Asia/Bishkek'

ASIA_BRUNEI = 'Asia/Brunei'

ASIA_CHITA = 'Asia/Chita'

ASIA_CHOIBALSAN = 'Asia/Choibalsan'

ASIA_COLOMBO = 'Asia/Colombo'

ASIA_DAMASCUS = 'Asia/Damascus'

ASIA_DHAKA = 'Asia/Dhaka'

ASIA_DILI = 'Asia/Dili'

ASIA_DUBAI = 'Asia/Dubai'

ASIA_DUSHANBE = 'Asia/Dushanbe'

ASIA_FAMAGUSTA = 'Asia/Famagusta'

ASIA_GAZA = 'Asia/Gaza'

ASIA_HEBRON = 'Asia/Hebron'

ASIA_HONG_KONG = 'Asia/Hong_Kong'

ASIA_HOVD = 'Asia/Hovd'

ASIA_HO_CHI_MINH = 'Asia/Ho_Chi_Minh'

ASIA_IRKUTSK = 'Asia/Irkutsk'

ASIA_JAKARTA = 'Asia/Jakarta'

ASIA_JAYAPURA = 'Asia/Jayapura'

ASIA_JERUSALEM = 'Asia/Jerusalem'

ASIA_KABUL = 'Asia/Kabul'

ASIA_KAMCHATKA = 'Asia/Kamchatka'

ASIA_KARACHI = 'Asia/Karachi'

ASIA_KATHMANDU = 'Asia/Kathmandu'

ASIA_KHANDYGA = 'Asia/Khandyga'

ASIA_KOLKATA = 'Asia/Kolkata'

ASIA_KRASNOYARSK = 'Asia/Krasnoyarsk'

ASIA_KUALA_LUMPUR = 'Asia/Kuala_Lumpur'

ASIA_KUCHING = 'Asia/Kuching'

ASIA_MACAU = 'Asia/Macau'

ASIA_MAGADAN = 'Asia/Magadan'

ASIA_MAKASSAR = 'Asia/Makassar'

ASIA_MANILA = 'Asia/Manila'

ASIA_NICOSIA = 'Asia/Nicosia'

ASIA_NOVOKUZNETSK = 'Asia/Novokuznetsk'

ASIA_NOVOSIBIRSK = 'Asia/Novosibirsk'

```
ASIA_OMSK = 'Asia/Omsk'

ASIA_ORAL = 'Asia/Oral'

ASIA_PONTIANAK = 'Asia/Pontianak'

ASIA_PYONGYANG = 'Asia/Pyongyang'

ASIA_QATAR = 'Asia/Qatar'

ASIA_QYZYLORDA = 'Asia/Qyzylorda'

ASIA_RIYADH = 'Asia/Riyadh'

ASIA_SAKHALIN = 'Asia/Sakhalin'

ASIA_SAMARKAND = 'Asia/Samarkand'

ASIA_SEOUL = 'Asia/Seoul'

ASIA_SHANGHAI = 'Asia/Shanghai'

ASIA_SINGAPORE = 'Asia/Singapore'

ASIA_SREDNEKOLYMSK = 'Asia/Srednekolymsk'

ASIA_TAIPEI = 'Asia/Taipei'

ASIA_TASHKENT = 'Asia/Tashkent'

ASIA_TBILISI = 'Asia/Tbilisi'

ASIA_TEHRAN = 'Asia/Tehran'

ASIA_THIMPHU = 'Asia/Thimphu'

ASIA_TOKYO = 'Asia/Tokyo'

ASIA_TOMSK = 'Asia/Tomsk'

ASIA_ULAANBAATAR = 'Asia/Ulaanbaatar'

ASIA_URUMQI = 'Asia/Urumqi'

ASIA_UST_NERA = 'Asia/Ust-Nera'

ASIA_VLADIVOSTOK = 'Asia/Vladivostok'

ASIA_YAKUTSK = 'Asia/Yakutsk'

ASIA_YANGON = 'Asia/Yangon'

ASIA_YEKATERINBURG = 'Asia/Yekaterinburg'

ASIA_YEREVAN = 'Asia/Yerevan'

ATLANTIC_AZORES = 'Atlantic/Azores'

ATLANTIC_BERMUDA = 'Atlantic/Bermuda'

ATLANTIC_CANARY = 'Atlantic/Canary'

ATLANTIC_CAPE_VERDE = 'Atlantic/Cape_Verde'

ATLANTIC_FAROE = 'Atlantic/Faroe'

ATLANTIC_MADEIRA = 'Atlantic/Madeira'

ATLANTIC_REYKJAVIK = 'Atlantic/Reykjavik'

ATLANTIC_SOUTH_GEORGIA = 'Atlantic/South_Georgia'
```

```
ATLANTIC_STANLEY = 'Atlantic/Stanley'

AUSTRALIA_ADELAIDE = 'Australia/Adelaide'

AUSTRALIA_BRISBANE = 'Australia/Brisbane'

AUSTRALIA_BROKEN_HILL = 'Australia/Broken_Hill'

AUSTRALIA_CURRIE = 'Australia/Currie'

AUSTRALIA_DARWIN = 'Australia/Darwin'

AUSTRALIA_EUCLA = 'Australia/Eucla'

AUSTRALIA_HOBART = 'Australia/Hobart'

AUSTRALIA_LINDEMAN = 'Australia/Lindeman'

AUSTRALIA_LORD_HOWE = 'Australia/Lord_Howe'

AUSTRALIA_MELBOURNE = 'Australia/Melbourne'

AUSTRALIA_PERTH = 'Australia/Perth'

AUSTRALIA_SYDNEY = 'Australia/Sydney'

EUROPE_AMSTERDAM = 'Europe/Amsterdam'

EUROPE_ANDORRA = 'Europe/Andorra'

EUROPE_ASTRAKHAN = 'Europe/Astrakhan'

EUROPE_ATHENS = 'Europe/Athens'

EUROPE_BELGRADE = 'Europe/Belgrade'

EUROPE_BERLIN = 'Europe/Berlin'

EUROPE_BRUSSELS = 'Europe/Brussels'

EUROPE_BUCHAREST = 'Europe/Bucharest'

EUROPE_BUDAPEST = 'Europe/Budapest'

EUROPE_CHISINAU = 'Europe/Chisinau'

EUROPE_COPENHAGEN = 'Europe/Copenhagen'

EUROPE_DUBLIN = 'Europe/Dublin'

EUROPE_GIBRALTAR = 'Europe/Gibraltar'

EUROPE_HELSINKI = 'Europe/Helsinki'

EUROPE_ISTANBUL = 'Europe/Istanbul'

EUROPE_KALININGRAD = 'Europe/Kaliningrad'

EUROPE_KIEV = 'Europe/Kiev'

EUROPE_KIROV = 'Europe/Kirov'

EUROPE_LISBON = 'Europe/Lisbon'

EUROPE_LONDON = 'Europe/London'

EUROPE_LUXEMBOURG = 'Europe/Luxembourg'

EUROPE_MADRID = 'Europe/Madrid'

EUROPE_MALTA = 'Europe/Malta'
```

```
EUROPE_MINSK = 'Europe/Minsk'

EUROPE_MONACO = 'Europe/Monaco'

EUROPE_MOSCOW = 'Europe/Moscow'

EUROPE_OSLO = 'Europe/Oslo'

EUROPE_PARIS = 'Europe/Paris'

EUROPE_PRAGUE = 'Europe/Prague'

EUROPE_RIGA = 'Europe/Riga'

EUROPE_ROME = 'Europe/Rome'

EUROPE_SAMARA = 'Europe/Samara'

EUROPE_SIMFEROPOL = 'Europe/Simferopol'

EUROPE_SOFIA = 'Europe/Sofia'

EUROPE_STOCKHOLM = 'Europe/Stockholm'

EUROPE_TALLINN = 'Europe/Tallinn'

EUROPE_TIRANE = 'Europe/Tirane'

EUROPE_ULYANOVSK = 'Europe/Ulyanovsk'

EUROPE_UZHGOROD = 'Europe/Uzhgorod'

EUROPE_VIENNA = 'Europe/Vienna'

EUROPE_VILNIUS = 'Europe/Vilnius'

EUROPE_VOLGOGRAD = 'Europe/Volgograd'

EUROPE_WARSAW = 'Europe/Warsaw'

EUROPE_ZAPOROZHYE = 'Europe/Zaporozhye'

EUROPE_ZURICH = 'Europe/Zurich'

INDIAN_CHAGOS = 'Indian/Chagos'

INDIAN_CHRISTMAS = 'Indian/Christmas'

INDIAN_COCOS = 'Indian/Cocos'

INDIAN_KERGUELEN = 'Indian/Kerguelen'

INDIAN_MAHE = 'Indian/Mahe'

INDIAN_MALDIVES = 'Indian/Maldives'

INDIAN_MAURITIUS = 'Indian/Mauritius'

INDIAN_REUNION = 'Indian/Reunion'

PACIFIC_APIA = 'Pacific/Apia'

PACIFIC_AUCKLAND = 'Pacific/Auckland'

PACIFIC_BOUGAINVILLE = 'Pacific/Bougainville'

PACIFIC_CHATHAM = 'Pacific/Chatham'

PACIFIC_CHUUK = 'Pacific/Chuuk'

PACIFIC_EASTER = 'Pacific/Easter'
```

PACIFIC_EFATE = 'Pacific/Efate'

PACIFIC_ENDERBURY = 'Pacific/Enderbury'

PACIFIC_FAKAOFO = 'Pacific/Fakaofo'

PACIFIC_FIJI = 'Pacific/Fiji'

PACIFIC_FUNAFUTI = 'Pacific/Funafuti'

PACIFIC_GALAPAGOS = 'Pacific/Galapagos'

PACIFIC_GAMBIER = 'Pacific/Gambier'

PACIFIC_GUADALCANAL = 'Pacific/Guadalcanal'

PACIFIC_GUAM = 'Pacific/Guam'

PACIFIC_HONOLULU = 'Pacific/Honolulu'

PACIFIC_KIRITIMATI = 'Pacific/Kiritimati'

PACIFIC_KOSRAE = 'Pacific/Kosrae'

PACIFIC_KWAJALEIN = 'Pacific/Kwajalein'

PACIFIC_MAJURO = 'Pacific/Majuro'

PACIFIC_MARQUESAS = 'Pacific/Marquesas'

PACIFIC_NAURU = 'Pacific/Nauru'

PACIFIC_NIUE = 'Pacific/Niue'

PACIFIC_NORFOLK = 'Pacific/Norfolk'

PACIFIC_NOUMEA = 'Pacific/Noumea'

PACIFIC_PAGO_PAGO = 'Pacific/Pago_Pago'

PACIFIC_PALAU = 'Pacific/Palau'

PACIFIC_PITCAIRN = 'Pacific/Pitcairn'

PACIFIC_POHNPEI = 'Pacific/Pohnpei'

PACIFIC_PORT_MORESBY = 'Pacific/Port_Moresby'

PACIFIC_RAROTONGA = 'Pacific/Rarotonga'

PACIFIC_TAHITI = 'Pacific/Tahiti'

PACIFIC_TARAWA = 'Pacific/Tarawa'

PACIFIC_TONGATAPU = 'Pacific/Tongatapu'

PACIFIC_WAKE = 'Pacific/Wake'

PACIFIC_WALLIS = 'Pacific/Wallis'

delete()

Delete this customer from the associated Node.

classmethod from_dict(*node*, *attributes=None*)

Create a new Customer initialized by a specified dictionary of attributes

Return type *Customer*

Parameters

- **node** – the node of the customer

- **`attributes`** – a dictionary representing the attributes of the new Customer

Returns a new Customer object

`get_events`()
Get all the events associated to this Customer.

Return type list

Returns A list containing Events object associated to this Customer

`get_mutation_tracker`()
Get the mutation tracker for this customer

Return type dict

Returns the mutation tracker of this customer

`patch`()
Patch this customer in the associated node, updating his attributes with the modified ones.

`post`(*force_update=False*)
Post this Customer in the associated Node.

Parameters **`force_update`** – if it's True and the customer already exists in the node, patch the customer with the modified properties.

`put`()
Put this customer in the associated node, substituting all the old attributes with the ones in this Customer.

`to_dict`()
Convert this Customer in a dictionary containing his attributes.

Return type dict

Returns a new dictionary representing the attributes of this Customer

6.3 Query

6.3.1 Criterion

class `contacthub.models.query.criterion.`**`Criterion`**(*first_element*, *operator*, *second_element=None*)
Bases: `object`

Criterion class for applying a criteria to our queries. A criteria consists of three elements:

- the queried attributes, an EntityField element if the criteria has a simple operator, another Criteria otherwise
- an operator, simple or complex, for querying data
- a value or a list of value to perform the comparision if the criteria has a simple operator, another Criteria otherwise

This criteria is consumed by a QueryBuilder object for returning a new Query object containing a dictionary representing the query in Contacthub

class COMPLEX_OPERATORS
List of complex operators handled in Contacthub

`AND = 'and'`

`OPERATORS = ['and', 'or']`

OR = 'or'

class SIMPLE_OPERATORS
List of simple operators handled in Contacthub

BETWEEN = 'BETWEEN'

EQUALS = 'EQUALS'

GT = 'GT'

GTE = 'GTE'

IN = 'IN'

IS_NOT_NULL = 'IS_NOT_NULL'

IS_NULL = 'IS_NULL'

LT = 'LT'

LTE = 'LTE'

NOT_EQUALS = 'NOT_EQUALS'

NOT_IN = 'NOT_IN'

OPERATORS = ['EQUALS', 'NOT_EQUALS', 'GT', 'GTE', 'LT', 'LTE', 'IN', 'NOT_IN', 'IS_

6.3.2 EntityField

class `contacthub.models.query.entity_field.`**`EntityField`**(*entity*, *field*)
Bases: `object`

Class for creating Critierion object for queries. Use this class for handling all operation that you want to implements in queries syntax and create new Criterion with related operations.

6.3.3 EntityMeta

class `contacthub.models.query.entity_meta.`**`EntityMeta`**
Bases: `type`

Metaclass for Properties class. Use this metaclass to handling the __getattr__ method and returns a new EntityField. This metaclass is useful for creating query objects and optimize the querying syntax.

6.3.4 Query

class `contacthub.models.query.query.`**`Query`**(*node*, *entity*, *previous_query=None*)
Bases: `object`

Query object for applying the specified query in the APIs.

Use this class for interact with the DeclarativeAPIManager Layer or APIManagerLevel and return the queried as object or json format variables

all()
Get all queried data of an entity from the API

Returns a ReadOnly list with all object queried

filter(*criterion*)
Create a new API Like query for Contacthub APIs (JSON Format)

Parameters **criterion** – the Criterion object for fields for query data

Returns a Query object containing the JSON object representing a query for the APIs

6.3.5 Module contents

contacthub.models.query.**between_**(*attribute*, *value1*, *value2*)
In operation for check if a str representing a date or a datetime object is between value1 and value2

Parameters

- **attribute** – a date attribute to check if is between two values
- **value1** – the the lower extreme for compare the date attribute
- **value2** – the the upper extreme for compare the date attribute

Returns a new Criterion object representing the BETWEEN operation

contacthub.models.query.**in_**(*value*, *list_attribute*)
IN operation for check if a value is in a list attribute

Parameters

- **value** – the value to check if is in the given list
- **list_attribute** – a list attribute for checking the presence of the value

Returns a new Criterion object representing the IN operation between a list and a value

contacthub.models.query.**not_in_**(*value*, *list_attribute*)
NOT IN operation for check if a value is NOT in a list attribute

Parameters

- **value** – the value to check if is not in the given list
- **list_attribute** – a list attribute for checking the absence of the value

Returns a new Criterion object representing the NOT IN operation between a list and a value

6.4 Like

class contacthub.models.like.**Like**(*customer*, *parent_attr=None*, *properties_class=None*, ***attributes*)
Bases: object

Like entiti definition

delete()
Remove this Like from the list of the Like for a Customer(specified in the constructor of the Like)

Returns a Like object representing the deleted Like

classmethod from_dict(*customer*, *attributes=None*, *parent_attr=None*, *properties_class=None*)
Create a new Like initialized by a specified dictionary of attributes

Parameters

- **customer** – the customer associated to this Like object

- **parent_attr** – the parent attribute for compiling the mutation tracker dictionary
- **attributes** – key-value arguments for generating the structure of the Like's attributes
- **properties_class** – reference to the actual Properties class

Returns a new Like object

post()

Post this Like in the list of the Like for a Customer(specified in the constructor of the Like)

Returns a Like object representing the posted Like

put()

Put this Like in the list of the Like for a Customer(specified in the constructor of the Like)

Returns a Like object representing the putted Like

to_dict()

Convert this Like in a dictionary containing his attributes.

Returns a new dictionary representing the attributes of this Like

6.5 Job

class contacthub.models.job.**Job**(*customer*, *parent_attr=None*, *properties_class=None*, ***attributes*)

Bases: object

Job entity definition.

delete()

Remove this Job from the list of the Job for a Customer(specified in the constructor of the Job)

Returns a Job object representing the deleted Job

classmethod from_dict(*customer*, *attributes=None*, *parent_attr=None*, *properties_class=None*)

Create a new Job initialized by a specified dictionary of attributes

Parameters

- **customer** – the customer associated to this Job object
- **parent_attr** – the parent attribute for compiling the mutation tracker dictionary
- **attributes** – key-value arguments for generating the structure of the Job's attributes
- **properties_class** – reference to the actual Properties class

Returns a new Job object

post()

Post this Job in the list of the Job for a Customer(specified in the constructor of the Job)

Returns a Job object representing the posted Job

put()

Put this Job in the list of the Job for a Customer(specified in the constructor of the Job)

Returns a Job object representing the putted Job

to_dict()

Convert this Job in a dictionary containing his attributes.

Returns a new dictionary representing the attributes of this Job

6.6 Education

class contacthub.models.education.**Education**(*customer*, *parent_attr=None*, *properties_class=None*, ***attributes*)

Bases: `object`

Education entity definition.

class SCHOOL_TYPES

Subclasses with school types for the schoolType field of Education

COLLEGE = 'COLLEGE'

HIGH_SCHOOL = 'HIGH_SCHOOL'

OTHER = 'OTHER'

PRIMARY_SCHOOL = 'PRIMARY_SCHOOL'

SECONDARY_SCHOOL = 'SECONDARY_SCHOOL'

delete()

Remove this Education from the list of the Education for a Customer(specified in the constructor of the Education)

Returns a Education object representing the deleted Education

classmethod from_dict(*customer*, *attributes=None*, *parent_attr=None*, *properties_class=None*)

Create a new Education initialized by a specified dictionary of attributes

Parameters

- **customer** – the customer associated to this education object
- **parent_attr** – the parent attribute for compiling the mutation tracker dictionary
- **attributes** – key-value arguments for generating the structure of the Education's attributes
- **properties_class** – reference to the actual Properties class

Returns a new Education object

post()

Post this Education in the list of the Education for a Customer(specified in the constructor of the Education)

Returns a Education object representing the posted Education

put()

Put this Education in the list of the Education for a Customer(specified in the constructor of the Education)

Returns a Education object representing the putted Education

to_dict()

Convert this Education in a dictionary containing his attributes.

Returns a new dictionary representing the attributes of this Education

6.7 Properties

class contacthub.models.properties.**Properties**(*parent=None*, *parent_attr=None*, ***attributes*)

Bases: `object`

Generic Properties for all entities

classmethod from_dict(*parent=None*, *parent_attr=None*, *attributes=None*)
Create a new Properties initialized by a specified dictionary of attributes

Parameters

- **parent** – the parent that generate this Properties object
- **parent_attr** – the parent attribute for compiling the mutation tracker dictionary
- **attributes** – key-value arguments for generating the structure of the Education's attributes

Returns a new Properties object

to_dict()
Convert this Properties object in a dictionary containing his attributes.

Returns a new dictionary representing the attributes of this Properties

contacthub.models.properties.**update_tracking_with_new_prop**(*mutations*, *new_properties*)
Add at the mutation tracker the new properties assigned with the setattr at a Properties object

Parameters

- **mutations** – the dictionary representing the mutation tracker
- **new_properties** – the properties to check for adding new attributes to the mutation tracker

6.8 Event

class contacthub.models.event.**Event**(*node*, ***attributes*)
Bases: object

Event entity definition

class CONTEXTS
‘ Event subclass for *context* field of Event. Choose one of the above API validated contexts or add your own in case of new types.

CONTACT_CENTER = 'CONTACT_CENTER'

DIGITAL_CAMPAIGN = 'DIGITAL_CAMPAIGN'

ECOMMERCE = 'ECOMMERCE'

IOT = 'IOT'

MOBILE = 'MOBILE'

OTHER = 'OTHER'

RETAIL = 'RETAIL'

SOCIAL = 'SOCIAL'

WEB = 'WEB'

class MODES

ACTIVE = 'ACTIVE'

PASSIVE = 'PASSIVE'

class TYPES

Event subclass for type field of Event. Choose one of the above API validated types or add your own in case of new types.

ABANDONED_CART = 'abandonedCart'

ABANDONED_SESSION = 'abandonedSession'

ADDED_COMPARE = 'addedCompare'

ADDED_PRODUCT = 'addedProduct'

ADDED_WISH_LIST = 'addedWishList'

CAMPAIGN_BLACKLISTED = 'campaignBlacklisted'

CAMPAIGN_BOUNCED = 'campaignBounced'

CAMPAIGN_LINK_CLICKED = 'campaignLinkClicked'

CAMPAIGN_MARKED_SPAM = 'campaignMarkedSpam'

CAMPAIGN_OPENED = 'campaignOpened'

CAMPAIGN_SENT = 'campaignSent'

CAMPAIGN_SUBSCRIBED = 'campaignSubscribed'

CAMPAIGN_UNSUBSCRIBED = 'campaignUnsubscribed'

CHANGED_SETTING = 'changedSetting'

CLICKED_LINK = 'clickedLink'

CLOSED_TICKET = 'closedTicket'

COMPLETED_ORDER = 'completedOrder'

EVENT_INVITED = 'eventInvited'

EVENT_PARTECIPATED = 'eventParticipated'

FORM_COMPILED = 'formCompiled'

GENERIC_ACTIVE_EVENT = 'genericActiveEvent'

GENERIC_PASSIVE_EVENT = 'genericPassiveEvent'

LOGGED_IN = 'loggedIn'

LOGGED_OUT = 'loggedOut'

OPENED_TICKET = 'openedTicket'

ORDER_SHIPPED = 'orderShipped'

REMOVED_COMPARE = 'removedCompare'

REMOVED_PRODUCT = 'removedProduct'

REMOVED_WISHLIST = 'removedWishList'

REPLIED_TICKET = 'repliedTicket'

REVIEWED_PRODUCT = 'reviewedProduct'

SEARCHED = 'searched'

SERVICE_SUBSCRIBED = 'serviceSubscribed'

SERVICE_UNSUBSCRIBED = 'serviceUnsubscribed'

VIEWED_PAGE = 'viewedPage'

VIEWED_PRODUCT = 'viewedProduct'

VIEWED_PRODUCT_CATEGORY = 'viewedProductCategory'

classmethod from_dict(*node*, *attributes=None*)
Create a new Properties initialized by a specified dictionary of attributes

Parameters

- **node** – the node of the customer supposed to be associated with this event
- **attributes** – key-value arguments for generating the structure of the Education's attributes

Returns a new Properties object

post()
Post this Event in the associated Node. For posting it and associate with a known customer, specify the customer id in the attributes of the Event. For associate it to an external Id or a session id of a customer, specify in the bringBackProperties object like: {'type':'EXTERNAL_ID', 'value':'value', 'nodeId':'nodeId' }

to_dict()
Convert this Event in a dictionary containing his attributes.

Returns a new dictionary representing the attributes of this Event

6.9 Errors

6.9.1 ApiError

exception `contacthub.errors.api_error.`**APIError**(**args*, ***kwargs*)
Bases: `requests.exceptions.HTTPError`

Class for API response error

6.9.2 OperationNotPermitted

exception `contacthub.errors.operation_not_permitted.`**OperationNotPermitted**
Bases: `exceptions.Exception`

Exception for blocking forbidden operations on entities.

6.10 Lib

6.10.1 ReadOnlyList

class `contacthub.lib.read_only_list.`**ReadOnlyList**
Bases: `list`

Read only list for managing list in the entities. This class blocks some possible operations on a regular list

static append(**args*, ***kwargs*)
Raise a new ValueError for blocking forbidden operations.

static extend(**args*, ***kwargs*)
Raise a new ValueError for blocking forbidden operations.

static insert(**args*, ***kwargs*)
Raise a new ValueError for blocking forbidden operations.

static not_implemented(**args*, ***kwargs*)
Raise a new ValueError for blocking forbidden operations.

static pop(**args*, ***kwargs*)
Raise a new ValueError for blocking forbidden operations.

static remove(**args*, ***kwargs*)
Raise a new ValueError for blocking forbidden operations.

static reverse(**args*, ***kwargs*)
Raise a new ValueError for blocking forbidden operations.

6.10.2 PaginatedList

class `contacthub.lib.paginated_list.`**PaginatedList**(*node*, *function*, *entity_class*, ***kwargs*)
Bases: `contacthub.lib.read_only_list.ReadOnlyList`

next_page()
Retrieve the next page of entities in this PaginatedList

Returns a PaginatedList containing the next page of entities compared to the current one

previous_page()
Retrieve the previous page of entities in this PaginatedList

Returns a PaginatedList containing the previous page of entities compared to the current one

6.10.3 utils

class `contacthub.lib.utils.`**DateEncoder**(*skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *encoding='utf-8'*, *default=None*)
Bases: `json.encoder.JSONEncoder`

Class for JSON encoding datetime and date object.

default(*obj*)
Serialize the given obj for JSON.

Parameters **obj** – the object to serialize

Returns a string ISO 8601 formatted

`contacthub.lib.utils.`**convert_properties_obj_in_prop**(*properties*, *properties_class*)
Convert the internal properties of *properties_class* object in a dictionary.

Parameters

- **properties** – the properties of a *properties_class* object to convert
- **properties_class** – if the internal properties of an object are instance of this class, we assign at the property the internal attributes instead of the object

`contacthub.lib.utils.`**`generate_mutation_tracker`**(*old_attributes*, *new_attributes*)

Given old attributes of an entity and the new attributes in a Properties object, this method creates a new dictionary based on the whole old attributes dictionary, with:

- the attributes in old_attributes updated with the attributes in new_attributes
- the attributes not in new_attributes (deleted) setted to None

Parameters

- **`old_attributes`** – The old attributes of an entity for create a mutation tracker dict updated
- **`new_attributes`** – The new attributes of an entity for create a mutation tracker dict updated

Returns a dictionary with the mutation between old_attributes and new_attributes

`contacthub.lib.utils.`**`get_dictionary_paths`**(*d*, *main_list*)

Set the given main_list with lists containing all the key-paths of a dictionary. For example: The key-paths for this list {a{b:{c:1, d:2}, e:3}} are a,b,c; a,b,d; a,e

Parameters

- **`d`** – the dictionary for gaining all the depth path
- **`main_list`** – a list for creating al the lists containing the paths of the dictt

`contacthub.lib.utils.`**`remove_empty_attributes`**(*body*)

`contacthub.lib.utils.`**`resolve_mutation_tracker`**(*mutation_tracker*)

From a dictionary with comma separated keys (e.g. {'a.b.c.d': 'e'}), creates a new dictionary with nested dictionaries, each of which containing a key taken from comma separated keys. (e.g. {'a': {'b' {'c': {'d': 'e'}}}})

Parameters **`mutation_tracker`** – the dictionary with comma separated keys

Returns a new dictionary containing nested dictionaries with old comma separated keys

Python Module Index

Index