# python-intercom Documentation

***Release 3.1.0***

**John Keyes**

**Oct 24, 2017**

# Contents

CHAPTER 1

# Installation

The simplest way to install python-intercom is with pip:

```
pip install python-intercom
```

or you may download a *.tar.gz* source archive from pypi:

```
tar xf python-intercom.tar.gz
cd python-intercom
python setup.py install
```

CHAPTER 2

# FAQ

## How do I start a session?

```
user = intercom.users.create(email='bingo@example.com')
# register a new session
user.new_session = True
intercom.users.save(user)
```

CHAPTER 3

# Changelog

- **3.1.0**
  - Added support for the Scroll API. (#156)
- **3.0.5**
  - Increased default request timeout to 90 seconds. This can also be set by the *INTERCOM_REQUEST_TIMEOUT* environment variable. (#154)
- **3.0.4**
  - Added *resource_type* attribute to lightweight classes. (#153)
- **3.0.3**
  - Removed *count* API operation, this is supported via *client.counts* now. (#152)
- **3.0.2**
  - Added multipage support for Event.find_all. (#147)
- **3.0.1**
  - Added support for HTTP keep-alive. (#146)
- 3.0
- **3.0b4**
  - Added conversation.mark_read method. (#136)
- **3.0b3**
  - Added TokenUnauthorizedError. (#134)
  - Added UTC datetime everywhere. (#130)
  - Fixed connection error when paginating. (#125)
  - Added Personal Access Token support. (#123)
  - Fixed links to Intercom API documentation. (#115)

- **3.0b2**
  - Added support for Leads. (#113)
  - Added support for Bulk API. (#112)
- **3.0b1**
  - Moved to new client based approach. (#108)
- **2.1.1**
  - No runtime changes.
- **2.1.0**
  - Adding interface support for opens, closes, and assignments of conversations. (#101)
  - Ensuring identity_hash only contains variables with valid values. (#100)
  - Adding support for unique_user_constraint and parameter_not_found errors. (#97)
- **2.0.0**
  - Added support for non-ASCII character sets. (#86)
  - Fixed response handling where no encoding is specified. (#81)
  - Added support for *None* values in *FlatStore*. (#88)
- **2.0.beta**
  - Fixed *UnboundLocalError* in *Request.parse_body*. (#72)
  - Added support for replies with an empty body. (#72)
  - Fixed a bug in identifying changed attributes when creating new resources. (#77)
- **2.0.alpha**
  - Added support for Intercom API v2.
  - Added support for Python 3.
- **0.2.13**
  - Fixed wildcard import from *intercom*. (#28)
- **0.2.12**
  - Added RTD theme to requirements.txt
- **0.2.11**
  - Added support for events. (#25)
  - Using RTD theme for documentation.
  - Fixed links to Intercom API docs.
- **0.2.10**
  - Added basic support for companies. (#18)
  - Fixed User.delete. (#19)
  - Fixed links to Intercom API docs.
  - Fixed doctests.
- **0.2.9**

  - Added *unsubscribed_from_emails* attribute to *User* object. (#15)
  - Added support for *last_request_at* parameter in *Intercom.create_user*. (#16)
  - Added support for page, per_page, tag_id, and tag_name parameters on *Intercom.get_users*. (#17)
- **0.2.8**
  - Added support for tagging. (#13)
  - Fixed installation into a clean python environment. (#12)
  - Fixed doctest.
  - Updated PEP8 formatting.
- **0.2.7**
  - Fixed delete user support to send bodyless request.
  - Added support for user notes.
- **0.2.6**
  - Added support for delete user.
- **0.2.5**
  - Fixed consistent version numbering (docs and code).
- **0.2.4**
  - Fixed handling of invalid JSON responses.
  - Fixed doctests to pass with current Intercom dummy API.
- **0.2.3**
  - Fixed version number of distribution to match documentation.
- **0.2.2**
  - Updated docstrings and doctests.
- **0.2.1**
  - Added some docstrings.
- **0.2**
  - Created source distribution. (#2)
  - Fixed error names. (#1)
- **0.1**
  - Initial release.

CHAPTER 4

# Development

## Running the tests

Run the unit tests:

```
nosetests tests/unit
```

Run the integration tests:

```
# THESE SHOULD ONLY BE RUN ON A TEST APP!
INTERCOM_PERSONAL_ACCESS_TOKEN=xxx nosetests tests/integration
```

## Generate the Documentation

```
cd docs
make html
```

## Code coverage

Generate a code coverage report:

```
nosetests --with-coverage --cover-package=intercom tests/unit
```

## Runtime Dependencies

- Requests – an HTTP library "for human beings"

- inflection – Inflection is a string transformation library. It singularizes and pluralizes English words, and transforms strings from CamelCase to underscored string.
- six – Six is a Python 2 and 3 compatibility library. It provides utility functions for smoothing over the differences between the Python versions with the goal of writing Python code that is compatible on both Python versions.
- certifi – Certifi is a carefully curated collection of Root Certificates for validating the trustworthiness of SSL certificates while verifying the identity of TLS hosts.
- pytz – pytz brings the Olson tz database into Python. This library allows accurate and cross platform timezone calculations. It also solves the issue of ambiguous times at the end of daylight saving time.

## Development Dependencies

- nose – makes unit testing easier.
- coverage – code coverage.
- mock – patching methods for unit testing.
- Sphinx – documentation decorator.
- Sphinx theme for readthedocs.org – theme for the documentation.

## Authors

python-intercom is written and maintained by John Keyes and various contributors:

### Development Lead

- John Keyes <john@keyes.ie> @jkeyes

### Patches and Suggestions

- vrachnis
- sdorazio
- Cameron Maske
- Martin-Zack Mekkaoui
- Marsel Mavletkulov
- Grant McConnaughey
- Robert Elliott
- Jared Morse
- Rafael
- jacoor
- maiiku
- Piotr Kilczuk

- Forrest Scofield
- Jordan Feldstein
- François Voron
- Gertjan Oude Lohuis

### Intercom

- Darragh Curran
- Bill de hÓra
- Jeff Gardner

CHAPTER 5

# intercom

## intercom package

### Subpackages

#### intercom.api_operations package

#### Submodules

#### intercom.api_operations.all module

Operation to retrieve all instances of a particular resource.

**class** intercom.api_operations.all.**All**
Bases: object

A mixin that provides *all* functionality.

**all**()
Return a CollectionProxy for the resource.

#### intercom.api_operations.count module

#### intercom.api_operations.delete module

Operation to delete an instance of a particular resource.

**class** intercom.api_operations.delete.**Delete**
Bases: object

A mixin that provides *delete* functionality.

**delete**(*obj*)
Delete the specified instance of this resource.

### intercom.api_operations.find module

Operation to find an instance of a particular resource.

**class** intercom.api_operations.find.**Find**
Bases: object

A mixin that provides *find* functionality.

**find**(**params*)
Find the instance of the resource based on the supplied parameters.

### intercom.api_operations.find_all module

Operation to find all instances of a particular resource.

**class** intercom.api_operations.find_all.**FindAll**
Bases: object

A mixin that provides *find_all* functionality.

**find_all**(**params*)
Find all instances of the resource based on the supplied parameters.

**proxy_class**
alias of CollectionProxy

### intercom.api_operations.load module

Operation to load an instance of a particular resource.

**class** intercom.api_operations.load.**Load**
Bases: object

A mixin that provides *load* functionality.

**load**(*resource*)
Load the resource from the latest data in Intercom.

### intercom.api_operations.save module

Operation to create or save an instance of a particular resource.

**class** intercom.api_operations.save.**Save**
Bases: object

A mixin that provides *create* and *save* functionality.

**create**(**params*)
Create an instance of the resource from the supplied parameters.

**id_present**(*obj*)
Return whether the obj has an *id* attribute with a value.

**identity_hash**(*obj*)
Return the identity_hash for this object.

**posted_updates**(*obj*)
Return whether the updates to this object have been posted to Intercom.

**save**(*obj*)
Save the instance of the resource.

### Module contents

Package for operations that can be performed on a resource.

### intercom.extended_api_operations package

### Submodules

### intercom.extended_api_operations.reply module

### intercom.extended_api_operations.users module

Operation to return all users for a particular Company.

**class** intercom.extended_api_operations.users.**Users**
Bases: object

A mixin that provides *users* functionality to Company.

**users**(*id*)
Return a CollectionProxy to all the users for the specified Company.

### Module contents

### intercom.generic_handlers package

### Submodules

### intercom.generic_handlers.base_handler module

### intercom.generic_handlers.count module

### intercom.generic_handlers.tag module

### intercom.generic_handlers.tag_find_all module

### Module contents

### intercom.lib package

### Submodules

### intercom.lib.flat_store module

**class** intercom.lib.flat_store.**FlatStore**(**args*, ***kwargs*)
Bases: dict

**setdefault**(*key*, *value=None*)

**update**(**args*, ***kwargs*)

### intercom.lib.setter_property module

**class** intercom.lib.setter_property.**SetterProperty**(*func*, *doc=None*)
Bases: object

### intercom.lib.typed_json_deserializer module

**class** intercom.lib.typed_json_deserializer.**JsonDeserializer**(*json*)
Bases: object

**deserialize**()

**deserialize_collection**(*collection_json*)

**deserialize_object**(*object_json*)

### Module contents

### intercom.traits package

### Submodules

### intercom.traits.api_resource module

**class** intercom.traits.api_resource.**Resource**(*_self*, **args*, ***params*)
Bases: object

**attributes**

**changed_attributes** = []

**client** = None

classmethod **from_api**(*response*)

**from_dict**(*dict*)

**from_response**(*response*)

**submittable_attribute**(*name*, *value*)

**to_dict**()

intercom.traits.api_resource.**custom_attribute_field**(*attribute*)

intercom.traits.api_resource.**datetime_value**(*value*)

intercom.traits.api_resource.**timestamp_field**(*attribute*)

intercom.traits.api_resource.**to_datetime_value**(*value*)

intercom.traits.api_resource.**type_field**(*attribute*)

intercom.traits.api_resource.**typed_value**(*value*)

#### intercom.traits.incrementable_attributes module

**class** intercom.traits.incrementable_attributes.**IncrementableAttributes**
Bases: object

**increment**(*key*, *value=1*)

#### Module contents

### Submodules

## intercom.admin module

**class** intercom.admin.**Admin**(*_self*, **args*, ***params*)
Bases: *intercom.traits.api_resource.Resource*

## intercom.collection_proxy module

**class** intercom.collection_proxy.**CollectionProxy**(*client*, *collection_cls*, *collection*, *finder_url*, *finder_params={}*)
Bases: six.Iterator

**extract_next_link**(*response*)

**get_first_page**()

**get_next_page**()

**get_page**(*url*, *params={}*)

**paging_info_present**(*response*)

## intercom.company module

**class** intercom.company.**Company**(*_self*, **args*, ***params*)
Bases: *intercom.traits.api_resource.Resource*

**flat_store_attributes**

**identity_vars = ['id', 'company_id']**

**update_verb = 'post'**

## intercom.conversation module

Collection module for Conversations.

**class** intercom.conversation.**Conversation**(*_self*, **args*, ***params*)
Bases: *intercom.traits.api_resource.Resource*

Collection class for Converations.

## intercom.count module

Count Resource.

**class** intercom.count.**Count**(*_self*, **args*, ***params*)
Bases: *intercom.traits.api_resource.Resource*

Collection class for Counts.

## intercom.errors module

**exception** intercom.errors.**ArgumentError**
Bases: exceptions.ValueError, *intercom.errors.IntercomError*

**exception** intercom.errors.**AuthenticationError**(*message=None*, *context=None*)
Bases: *intercom.errors.IntercomError*

**exception** intercom.errors.**BadGatewayError**(*message=None*, *context=None*)
Bases: *intercom.errors.IntercomError*

**exception** intercom.errors.**BadRequestError**(*message=None*, *context=None*)
Bases: *intercom.errors.IntercomError*

**exception** intercom.errors.**HttpError**(*message=None*, *context=None*)
Bases: *intercom.errors.IntercomError*

**exception** intercom.errors.**IntercomError**(*message=None*, *context=None*)
Bases: exceptions.Exception

**exception** intercom.errors.**MultipleMatchingUsersError**(*message=None*, *context=None*)
Bases: *intercom.errors.IntercomError*

**exception** intercom.errors.**RateLimitExceeded**(*message=None*, *context=None*)
Bases: *intercom.errors.IntercomError*

**exception** intercom.errors.**ResourceNotFound**(*message=None*, *context=None*)
Bases: *intercom.errors.IntercomError*

**exception** intercom.errors.**ResourceNotRestorable**(*message=None*, *context=None*)
Bases: *intercom.errors.IntercomError*

**exception** intercom.errors.**ServerError**(*message=None*, *context=None*)
Bases: *intercom.errors.IntercomError*

**exception** intercom.errors.**ServiceUnavailableError**(*message=None*, *context=None*)
Bases: *intercom.errors.IntercomError*

**exception** intercom.errors.**TokenNotFoundError**(*message=None*, *context=None*)
Bases: *intercom.errors.IntercomError*

**exception** intercom.errors.**TokenUnauthorizedError**(*message=None*, *context=None*)
Bases: *intercom.errors.IntercomError*

**exception** intercom.errors.**UnexpectedError**(*message=None*, *context=None*)
Bases: *intercom.errors.IntercomError*

## intercom.event module

**class** intercom.event.**Event**(*_self*, **args*, ***params*)
Bases: *intercom.traits.api_resource.Resource*

## intercom.message module

**class** intercom.message.**Message**(*_self*, **args*, ***params*)
Bases: *intercom.traits.api_resource.Resource*

## intercom.note module

**class** intercom.note.**Note**(*_self*, **args*, ***params*)
Bases: *intercom.traits.api_resource.Resource*

## intercom.notification module

**class** intercom.notification.**Notification**(*_self*, **args*, ***params*)
Bases: *intercom.traits.api_resource.Resource*

**load**

**model**

**model_type**

## intercom.request module

**class** intercom.request.**Request**(*http_method*, *path*, *http_session=None*)
Bases: object

**execute**(*base_url*, *auth*, *params*)

**message_for_unexpected_error_with_type**(*error_details*, *http_code*)

**message_for_unexpected_error_without_type**(*error_details*, *http_code*)

**parse_body**(*resp*)

**raise_application_errors_on_failure**(*error_list_details*, *http_code*)

**raise_errors_on_failure**(*resp*)

**send_request_to_path**(*base_url*, *auth*, *params=None*)
Construct an API request, send it to the API, and parse the response.

**set_rate_limit_details**(*resp*)

**timeout = 90**

**class** intercom.request.**ResourceEncoder**(*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

**default**(*o*)

intercom.request.**configure_timeout**()
Configure the request timeout.

## intercom.segment module

**class** intercom.segment.**Segment**(*_self*, **args*, ***params*)
Bases: *intercom.traits.api_resource.Resource*

## intercom.subscription module

**class** intercom.subscription.**Subscription**(*_self*, **args*, ***params*)
Bases: *intercom.traits.api_resource.Resource*

## intercom.tag module

**class** intercom.tag.**Tag**(*_self*, **args*, ***params*)
Bases: *intercom.traits.api_resource.Resource*

## intercom.user module

**class** intercom.user.**User**(*_self*, **args*, ***params*)
Bases: *intercom.traits.api_resource.Resource*, *intercom.traits.incrementable_attributes.IncrementableAttributes*

**flat_store_attributes**

**identity_vars = ['id', 'email', 'user_id']**

**update_verb = 'post'**

## intercom.utils module

intercom.utils.**constantize_singular_resource_name**(*resource_name*)

intercom.utils.**define_lightweight_class**(*resource_name*, *class_name*)
Return a lightweight class for deserialized payload objects.

intercom.utils.**entity_key_from_type**(*type*)

intercom.utils.**pluralize**(*str*)

intercom.utils.**resource_class_to_collection_name**(*cls*)

intercom.utils.**resource_class_to_name**(*cls*)

## Module contents

CHAPTER 6

# Installation

Stable releases of python-intercom can be installed with pip or you may download a *.tgz* source archive from pypi. See the *Installation* page for more detailed instructions.

If you want to use the latest code, you can grab it from our Git repository, or fork it.

CHAPTER 7

# Usage

## Authorization

Intercom documentation: Personal Access Tokens.

```
from intercom.client import Client
intercom = Client(personal_access_token='my_personal_access_token')
```

## Users

### Create or Update User

Intercom documentation: Create or Update Users.

```
intercom.users.create(user_id='1234', email='bob@example.com')
```

### Updating the Last Seen Time

Intercom documentation: Updating the Last Seen Time.

```
user = intercom.users.create(used_id='25', last_request_at=datetime.utcnow())
```

### List Users

Intercom documentation: List Users.

```
for user in intercom.users.all():
    ...
```

### List by Tag, Segment, Company

Intercom documentation: List by Tag, Segment, Company.

```
# tag request
intercom.users.find_all(tag_id='30126')

# segment request
intercom.users.find_all(segment_id='30126')
```

### View a User

Intercom documentation: View a User.

```
# ID request
intercom.users.find(id='1')

# User ID request
intercom.users.find(user_id='1')

# Email request
intercom.users.find(email='bob@example.com')
```

### Delete a User

Intercom documentation: Deleting a User.

```
# ID Delete Request
user = intercom.users.find(id='1')
deleted_user = intercom.users.delete(user)

# User ID Delete Request
user = intercom.users.find(user_id='1')
deleted_user = intercom.users.delete(user)

# Email Delete Request
user = intercom.users.find(email='bob@example.com')
deleted_user = intercom.users.delete(user)
```

## Companies

### Create or Update Company

Intercom documentation: Create or Update Company.

```
intercom.companies.create(company_id=6, name="Blue Sun", plan="Paid")
```

### List Companies

Intercom documentation: List Companies.

```
for company in intercom.companies.all():
    ...
```

### List by Tag or Segment

Intercom documentation: List by Tag or Segment.

```
# tag request
intercom.companies.find(tag_id="1234")

# segment request
intercom.companies.find(segment_id="4567")
```

### View a Company

Intercom documentation: View a Company.

```
intercom.companies.find(id="41e66f0313708347cb0000d0")
```

### List Company Users

Intercom documentation: List Company Users.

```
company = intercom.companies.find(id="41e66f0313708347cb0000d0")
for user in company.users:
    ...
```

## Admins

### List Admins

Intercom documentation: List Admins.

```
for admin in intercom.admins.all():
    ...
```

## Tags

### Create and Update Tags

Intercom documentation: Create and Update Tags.

```
# Create Request
tag = intercom.tags.create(name='Independentt')

# Update Request
intercom.tags.tag_users(name='Independent', id=tag.id)
```

### Tag or Untag Users & Companies

Intercom documentation: Tag or Untag Users & Companies.

```
# Multi-User Tag Request
intercom.tags.tag_users('Independent', ["42ea2f1b93891f6a99000427",
↪"42ea2f1b93891f6a99000428"])

# Untag Request
intercom.tags.untag_users('blue', ["42ea2f1b93891f6a99000427"])
```

### Delete a Tag

Intercom documentation: Delete a Tag.

```
intercom.tags.delete()
```

### List Tags for an App

Intercom Documentation: List Tags for an App.

```
for intercom.tags in Tag.all():
    ...
```

## Segments

### List Segments

Intercom Documentation: List Segments.

```
for segment in intercom.segments.all():
    ...
```

### View a Segment

Intercom Documentation: View a Segment.

```
intercom.segments.find(id='1234')
```

## Notes

### Create a Note

Intercom documentation: Create a Note.

```
intercom.notes.create(email="joe@exampe.com", body="Text for the note")
```

### List Notes for a User

Intercom documentation: List Notes for a User.

```
# User ID Request
for note in intercom.notes.find_all(user_id='123'):
    ...

# User Email Request
for note in intercom.notes.find_all(email='foo@bar.com'):
    ...
```

### View a Note

Intercom documentation: View a Note.

```
intercom.notes.find(id='1234')
```

## Events

### Submitting Events

Intercom documentation: Submitting Events.

```
intercom.events.create(event_name="Eventful 1", email=user.email, created_
↪at=1403001013)
```

## Counts

### Getting counts

Intercom documentation: Getting Counts.

```
# Conversation Admin Count
intercom.counts.for_type(type='conversation', count='admin')

# User Tag Count
intercom.counts.for_type(type='user', count='tag')

# User Segment Count
intercom.counts.for_type(type='user', count='segment')

# Company Tag Count
intercom.counts.for_type(type='company', count='tag')

# Company User Count
intercom.counts.for_type(type='company', count='user')

# Global App Counts
intercom.counts.for_type()
```

# Conversations

## Admin Initiated Conversation

Intercom documentation: Admin Initiated Conversation.

```
message_data = {
    'message_type': 'email',
    'subject': 'This Land',
    'body': "Har har har! Mine is an evil laugh!",
    'template': "plain",
    'from': {
        'type': "admin",
        'id': "394051"
    },
    'to': {
        'type': "user",
        'id': "536e564f316c83104c000020"
    }
}
intercom.messages.create(**message_data)
```

## User Initiated Conversation

Intercom documentation: User Initiated Conversation.

```
message_data = {
    'from': {
        'type': "user",
        'id': "536e564f316c83104c000020"
    },
    'body': "Hey"
}
intercom.messages.create(**message_data)
```

## List Conversations

Intercom documentation: List Conversations.

```
intercom.conversations.find_all(type='admin', id=25, open=True)
```

## Get a Single Conversation

Intercom documentation: Get a Single Conversation.

```
intercom.conversations.find(id='147')
```

## Replying to a Conversation

Intercom documentation: Replying to a Conversation.

```
conversation.reply(type='user', email='bob@example.com', message_type='comment', body=
↪'foo')
```

### Marking a Conversation as Read

Intercom documentation: Marking a Conversation as Read.

```
conversation.read = True
conversation.save()
```

## Webhooks and Notifications

### Manage Subscriptions

Intercom documentation: Manage Subscriptions.

```
intercom.subscriptions.create(service_type='web', url='http://example.com', topics=[
↪'all'])
```

### View a Subscription

Intercom documentation: View a Subscription.

```
intercom.subscriptions.find(id='123')
```

### List Subscriptions

Intercom documentation: List Subscriptions.

```
for subscription in intercom.subscriptions.all():
    ...
```

CHAPTER 8

# Development

Our *Development* page has detailed instructions on how to run our tests, and to produce coverage and pylint reports.

CHAPTER 9

# Changelog

The *Changelog* keeps track of changes per release.

# Python Module Index

## i

# Index

## T



## U