python-intercom Documentation

Release 0.2.13

John Keyes

May 12, 2015

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
```

intercom

2.1 intercom Package

2.1.1 event Module

Intercom API wrapper.

class intercom.events.**Event**
Bases: *intercom.user.UserId*

classmethod create(*event_name=None*, *user_id=None*, *email=None*, *metadata=None*)

event_name
The name of the Event.

save()
Create an Event from this objects properties:

```
>>> event = Event()
>>> event.event_name = "shared-item"
>>> event.email = "joe@example.com"
>>> event.save()
```

2.1.2 intercom Package

Intercom API wrapper.

exception intercom.__init__.**AuthenticationError**(*message*, *result=None*)
Bases: *intercom.intercom.IntercomError*

Raised when a request cannot be authenticated by the API.

exception intercom.__init__.**BadGatewayError**(*message*, *result=None*)
Bases: *intercom.intercom.IntercomError*

Raised when a request does not reach the API due to a 502.

class intercom.__init__.**Intercom**
Bases: object

Intercom API Wrapper

api_endpoint = 'https://api.intercom.io/v1/'

api_key = None

api_version = 1

app_id = None

classmethod create_event(*event_name=None*, *user_id=None*, *email=None*, *metadata=None*)
Create an event

classmethod create_impression(*user_id=None*, *email=None*, *user_ip=None*, *user_agent=None*, *location=None*)
Create an impression.

```
>>> result = Intercom.create_impression(email="somebody@example.com",
... user_agent="MyApp/1.0", user_ip="2.3.4.5")
>>> result['unread_messages']
1
```

classmethod create_message_thread(*user_id=None*, *email=None*, *body=None*)
Create a MessageThread.

```
>>> message_thread = Intercom.create_message_thread(
... email="somebody@example.com",
... body="Hey Intercom, What is up?")
>>> message_thread['thread_id']
5591
>>> len(message_thread['messages'])
3
>>> message_thread['messages'][0]['html']
u'<p>Hey Intercom, What is up?</p>
```

'

classmethod create_note(*user_id=None*, *email=None*, *body=None*)
Create a note.

```
>>> result = Intercom.create_note(email="somebody@example.com",
... body="This is a note")
>>> result['html']
u'<p>This is a note</p>'
>>> result['user']['email']
u'somebody@example.com'
```

classmethod create_tag(*name*, *tag_or_untag*, *user_ids=None*, *emails=None*)
Create a tag (and maybe tag users).

```
>>> tag = Intercom.create_tag("Free Trial", "tag",
... user_ids=["abc123", "def456"])
>>> tag['id'] != None
True
>>> tag['name']
u'Free Trial'
>>> tag['tagged_user_count']
2
```

classmethod create_user(***kwargs*)
Creates a user.

N.B. Social and geo location data is fetched asynchronously, so a secondary call to users will be required to fetch it.

Arguments

- `user_id`: required (if no email) — a unique string identifier for the user

- `email`: required (if no user_id) — the user's email address
- `name`: The user's full name
- `created_at`: A UNIX timestamp representing the date the user was created
- `custom_data`: A hash of key/value pairs containing any other data about the user you want Intercom to store.
- `last_seen_ip`: An ip address (e.g. "1.2.3.4") representing the last ip address the user visited your application from. (Used for updating location_data)
- `last_seen_user_agent`: The user agent the user last visited your application with.
- `companies`: An array of hashes describing the companies this user belongs to. Currently companies are not returned in the response.
- `last_request_at or last_impression_at`: A UNIX timestamp representing the date the user last visited your application.
- `unsubscribed_from_emails`: A boolean value representing the users unsubscribed status.

```
>>> user = Intercom.create_user(user_id='7902',
... email='ben@intercom.io',
... name='Somebody', created_at=1270000000, last_seen_ip='1.2.3.4',
... custom_data={ 'app_name': 'Genesis'}, last_request_at=1300000000)
>>> user['name']
u'Somebody'
>>> user['custom_data']['app_name']
u'Genesis'
>>> user['last_impression_at']
1300000000
```

classmethod `delete_user` (*user_id=None*, *email=None*)

Delete a user.

```
>>> user = Intercom.get_user(user_id='123')
>>> user['email']
u'somebody@example.com'
```

classmethod `get_message_threads` (*user_id=None*, *email=None*, *thread_id=None*)

If a thread_id is specified, this returns a specific MessageThread (if it can find one), otherwise it returns all MessageThreads for the particular user.

```
>>> message_threads = Intercom.get_message_threads(
... email="somebody@example.com")
>>> type(message_threads)
<type 'list'>
>>> message_thread = Intercom.get_message_threads(
... email="somebody@example.com", thread_id=5591)
>>> type(message_thread)
<type 'dict'>
```

classmethod `get_tag` (*name=None*)

Return a dict for the tag by the specified name.

```
>>> tag = Intercom.get_tag(name="Free Trial")
>>> tag['id'] != None
True
>>> tag['name']
u'Free Trial'
```

```
>>> tag['tagged_user_count']
2
```

classmethod get_user(*email=None*, *user_id=None*)
Return a dict for the user represented by the specified email or user_id.

```
>>> user = Intercom.get_user(user_id='123')
>>> user['name']
u'Somebody'
```

classmethod get_users(***kwargs*)
Returns a paginated list of all users in your application on Intercom.

Arguments

- `page`: optional (defaults to 1)
- `per_page`: optional (defaults to 500, max value of 500)
- `tag_id`: optional — query for users that are tagged with a specific tag.
- `tag_name`: optional — query for users that are tagged with a specific tag.

Response

- `users`: an array of User objects (same as returned by getting a single User)
- `total_count`: the total number of Users tracked in your Intercom application
- `page`: the current requested page
- `next_page`: the next page number, if any
- `previous_page`: the previous page number, if any
- `total_pages`: the total number of pages

```
>>> result = Intercom.get_users()
>>> type(result)
<type 'dict'>
>>> len(result['users'])
3
```

classmethod reply_message_thread(*user_id=None*, *email=None*, *thread_id=None*, *body=None*, *read=None*)
Reply to the specific thread.

```
>>> message_thread = Intercom.reply_message_thread(
... email="somebody@example.com",
... thread_id=5591,
... body="If you're not talking to me you must be talking to someone")
>>> len(message_thread)
9
>>> message_thread['thread_id']
5591
>>> len(message_thread['messages'])
3
```

timeout = 10

classmethod update_tag(*name*, *tag_or_untag*, *user_ids=None*, *emails=None*)
Update a tag (and maybe tag users).

```
>>> tag = Intercom.update_tag("Free Trial", "tag",
... user_ids=["abc123", "def456"])
>>> tag['id'] != None
True
>>> tag['name']
u'Free Trial'
>>> tag['tagged_user_count']
2
```

classmethod update_user(***kwargs*)
Update a user with the available parameters.

```
>>> user = Intercom.get_user(user_id='123')
>>> user['name']
u'Somebody'
>>> user = Intercom.update_user(user_id='123', name='Guido')
>>> user['name']
u'Guido'
```

exception intercom.__init__.**ResourceNotFound**(*message*, *result=None*)
Bases: *intercom.intercom.IntercomError*

Raised when a resource cannot be found e.g. a non-existant User.

exception intercom.__init__.**ServerError**(*message*, *result=None*)
Bases: *intercom.intercom.IntercomError*

Raised when the API returns an error other than an auth or not found.

exception intercom.__init__.**ServiceUnavailableError**(*message*, *result=None*)
Bases: *intercom.intercom.IntercomError*

Raised when the API cannot be handle a request.

class intercom.__init__.**Impression**
Bases: *intercom.user.UserId*

An Impression represents an interaction between a User and your application.

classmethod create(*user_id=None*, *email=None*, *user_ip=None*, *user_agent=None*, *location=None*)
Create an Impression.

```
>>> Impression.create(email="somebody@example.com",
...         location="/pricing/upgrade",
...         user_ip="1.2.3.4",
...         user_agent="my-service-iphone-app-1.2")
{u'unread_messages': 1}
```

location
The location where this Impression originated e.g. /pricing/upgrade or 'DesktopApp: Pricing' or 'iOS'.

save()
Create an Impression from this objects properties:

```
>>> impression = Impression()
>>> impression.email = "somebody@example.com"
>>> impression.location = "/pricing/upgrade"
>>> impression.user_ip = "1.2.3.4"
>>> impression.user_agent = "my-service-iphone-app-1.2"
>>> impression.save()
>>> impression.unread_messages
1
```

unread_messages
The number of unread messages for the User who made the Impression for the current location.

user_agent
The User Agent that created this Impression e.g. the browser User Agent, or the name and version of an application.

user_ip
The IP address where this Impression originated.

class intercom.__init__.**MessageThread**
Bases: dict

An Intercom conversation between an admin and a User.

body

classmethod create(*user_id=None*, *email=None*, *body=None*)
Creates a new converstion.

```
>>> email = "somebody@example.com"
>>> body = "Hi everybody, I'm Doctor Nick"
>>> message_thread = MessageThread.create(email=email, body=body)
>>> message_thread.thread_id
5591
>>> len(message_thread.messages)
3
```

created_at
Sets a timestamp of when the last update occurred.

classmethod find(*user_id=None*, *email=None*, *thread_id=None*)
Finds a particular conversation for a particular user.

```
>>> message_thread = MessageThread.find(email="somebody@example.com")
Traceback (most recent call last):
    ...
ValueError: No thread_id specified
>>> message_thread = MessageThread.find(email="somebody@example.com",
... thread_id=5591)
>>> len(message_thread.messages)
3
>>> message = message_thread.messages[0]
>>> type(message)
<class 'intercom.message_thread.Message'>
```

classmethod find_all(*user_id=None*, *email=None*)
Finds all Messages for a particular user.

```
>>> message_threads = MessageThread.find_all(
... email="somebody@example.com")
>>> len(message_threads)
1
```

messages
Returns a list of Messages in this MessageThread.

read
Returns whether this thread has been read or not.

classmethod reply(*user_id=None*, *email=None*, *thread_id=None*, *body=None*, *read=None*)
Reply to an existing conversation.

```
>>> email = "somebody@example.com"
>>> thread_id = 5591
>>> body = "Are you talking to me?"
>>> message_thread = MessageThread.reply(email=email,
... thread_id=thread_id, body=body)
>>> len(message_thread.messages)
3
>>> message_thread.messages[0].html
u'<p>Hey Intercom, What is up?</p>
```

** '**

```
>>> message_thread.messages[1].html
u'<p>Not much, you?
```

** '**

```
>>> message_thread.messages[2].html
u'<p>Not much either :(</p>
```

'

set_body(*value*)
Set the body of a reply.

thread_id
Returns the thread_id of this MessageThread.

updated_at
Returns a datetime of when the last update occurred.

class intercom.__init__.**Note**
Bases: *intercom.user.UserId*

A note on a User.

body
The body of the note.

classmethod create(*user_id=None*, *email=None*, *body=None*)
Create a Note.

```
>>> note = Note.create(email="somebody@example.com",
...         body="This is a note.")
>>> note.created_at.year
2011
>>> note.html
u'<p>This is a note</p>'
```

created_at
Returns the datetime this note was created – set by an API response.

html
Get the html of the note – set by an API response.

save()
Create a Note from this objects properties:

```
>>> note = Note()
>>> note.email = "somebody@example.com"
>>> note.body = "This is a note."
>>> note.save()
>>> note.html
u'<p>This is a note</p>'
```

user
Get the noted user – set by an API response.

class intercom.__init__.**User**
Bases: *intercom.user.UserId*

Object representing http://docs.intercom.io/#UserData).

classmethod all()
Return all of the Users.

```
>>> users = User.all()
>>> len(users)
3
>>> users[0].email
u'first.user@example.com'
```

attributes = ('user_id', 'email', 'name', 'created_at', 'custom_data', 'last_seen_ip', 'last_seen_user_agent', 'companie

companies
Get the companies of a user. Currently unsupported by the Intercom API so an AttributeError is raised.

```
>>> user = User(email="somebody@example.com")
>>> user.companies
Traceback (most recent call last):
    ...
AttributeError: companies is a write-only property
```

company
Get the company of a user. Currently unsupported by the Intercom API so an AttributeError is raised.

```
>>> user = User(email="somebody@example.com")
>>> user.company
Traceback (most recent call last):
    ...
AttributeError: company is a write-only property
```

classmethod create(***kwargs*)
Create or update a user.

```
>>> user = User.create(email="somebody@example.com",last_impression_at=1400000000)
>>> user.name
u'Somebody'
>>> user.last_impression_at.year
2011
```

created_at
Returns the datetime this User started using your application.

custom_data
Returns a CustomData object for this User.

```
>>> users = User.all()
>>> custom_data = users[0].custom_data
```

```
>>> type(custom_data)
<class 'intercom.user.CustomData'>
>>> custom_data['monthly_spend']
155.5
```

classmethod delete(*user_id=None*, *email=None*)
Deletes a user.

```
>>> user = User.delete(email="somebody@example.com")
>>> user.user_id
u'123'
>>> user = User.delete(user_id="123")
>>> user.email
u'somebody@example.com'
```

classmethod find(*user_id=None*, *email=None*)
Find a user by email or user_id.

```
>>> user = User.find(email="somebody@example.com")
>>> user.user_id
u'123'
>>> user.name
u'Somebody'
>>> user = User.find(user_id=123)
>>> user.email
u'somebody@example.com'
>>> user.name
u'Somebody'
```

classmethod find_by_email(*email*)
Find a user by email.

```
>>> user = User.find_by_email("somebody@example.com")
>>> user.user_id
u'123'
>>> user.name
u'Somebody'
```

classmethod find_by_user_id(*user_id*)
Find a user by user_id.

```
>>> user = User.find(user_id=123)
>>> user.email
u'somebody@example.com'
>>> user.name
u'Somebody'
```

last_impression_at
Returns the datetime this User last used your application.

last_request_at
Get last time this User interacted with your application.

last_seen_ip
Returns the last seen IP address.

last_seen_user_agent
Returns the last seen User Agent.

location_data
Returns a LocationData object for this User.

```
>>> users = User.all()
>>> location_data = users[0].location_data
>>> type(location_data)
<class 'intercom.user.LocationData'>
>>> location_data.country_name
u'Chile'
>>> location_data.city_name
u'Santiago'
```

name
Returns the name e.g. Joe Bloggs.

relationship_score
Returns the relationship score.

save()
Creates or updates a User.

```
>>> user = User()
>>> user.email = "somebody@example.com"
>>> user.save()
>>> user.name
u'Somebody'
```

session_count
Returns how many sessions this User has used on your application.

social_profiles
Returns a list of SocialProfile objects for this User.

```
>>> users = User.all()
>>> social_profiles = users[0].social_profiles
>>> len(social_profiles)
2
>>> type(social_profiles[0])
<class 'intercom.user.SocialProfile'>
>>> social_profiles[0].type
u'twitter'
>>> social_profiles[0].url
u'http://twitter.com/abc'
```

unsubscribed_from_emails
Returns whether or not the user has unsubscribed from emails

class intercom.__init__.**Tag**
Bases: dict

Represents a tag.

classmethod create(*name*, *tag_or_untag*, *user_ids=None*, *emails=None*)
Create a new tag an optionally tag user.

```
>>> tag = Tag.create(user_ids=["abc123", "def456"],
...         name="Free Trial", tag_or_untag="tag")
>>> tag.id != None
True
>>> tag.name
u'Free Trial'
>>> tag.tagged_user_count
2
```

emails
The emails of the tag.

classmethod find(***params*)
Find a tag using params.

```
>>> tag = Tag.find(name="Free Trial")
>>> tag.id != None
True
>>> tag.name
u'Free Trial'
>>> tag.tagged_user_count
2
```

classmethod find_by_name(*name*)
Find a tag by name.

```
>>> tag = Tag.find_by_name("Free Trial")
>>> tag.id != None
True
>>> tag.name
u'Free Trial'
>>> tag.tagged_user_count
2
```

id
The id of the tag.

name
Get the name of the tag.

save()
Update a tag:

```
>>> tag = Tag()
>>> tag.user_ids = ["abc123", "def456"]
>>> tag.name = "Free Trial"
>>> tag.tag_or_untag = "tag"
>>> tag.save()
>>> tag.tagged_user_count
2
```

segment
Get the segment of the tag.

tag_or_untag
The tag_or_untag of the tag.

tagged_user_count
Get the tagged_user_count of the tag.

user_ids
The user_ids of the tag.

class intercom.__init__.**Event**
Bases: *intercom.user.UserId*

classmethod create(*event_name=None*, *user_id=None*, *email=None*, *metadata=None*)

event_name
The name of the Event.

save()
Create an Event from this objects properties:

```
>>> event = Event()
>>> event.event_name = "shared-item"
>>> event.email = "joe@example.com"
>>> event.save()
```

2.1.3 `impression` Module

Impression module.

```
>>> from intercom import Intercom
>>> Intercom.app_id = 'dummy-app-id'
>>> Intercom.api_key = 'dummy-api-key'
>>> from intercom import Impression
```

class intercom.impression.**Impression**
Bases: *intercom.user.UserId*

An Impression represents an interaction between a User and your application.

classmethod create(*user_id=None*, *email=None*, *user_ip=None*, *user_agent=None*, *location=None*)
Create an Impression.

```
>>> Impression.create(email="somebody@example.com",
...         location="/pricing/upgrade",
...         user_ip="1.2.3.4",
...         user_agent="my-service-iphone-app-1.2")
{u'unread_messages': 1}
```

location
The location where this Impression originated e.g. /pricing/upgrade or 'DesktopApp: Pricing' or 'iOS'.

save()
Create an Impression from this objects properties:

```
>>> impression = Impression()
>>> impression.email = "somebody@example.com"
>>> impression.location = "/pricing/upgrade"
>>> impression.user_ip = "1.2.3.4"
>>> impression.user_agent = "my-service-iphone-app-1.2"
>>> impression.save()
>>> impression.unread_messages
1
```

unread_messages
The number of unread messages for the User who made the Impression for the current location.

user_agent
The User Agent that created this Impression e.g. the browser User Agent, or the name and version of an application.

user_ip
The IP address where this Impression originated.

2.1.4 `intercom` Module

intercom module

All of the API requests are created, and the API responses are parsed here.

```
>>> from intercom import Intercom
>>> Intercom.app_id = 'dummy-app-id'
>>> Intercom.api_key = 'dummy-api-key'
```

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

Raised when a request cannot be authenticated by the API.

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

Raised when a request does not reach the API due to a 502.

class intercom.intercom.**Intercom**
Bases: object

Intercom API Wrapper

api_endpoint = 'https://api.intercom.io/v1/'

api_key = None

api_version = 1

app_id = None

classmethod create_event(*event_name=None*, *user_id=None*, *email=None*, *metadata=None*)
Create an event

classmethod create_impression(*user_id=None*, *email=None*, *user_ip=None*, *user_agent=None*, *location=None*)
Create an impression.

```
>>> result = Intercom.create_impression(email="somebody@example.com",
... user_agent="MyApp/1.0", user_ip="2.3.4.5")
>>> result['unread_messages']
1
```

classmethod create_message_thread(*user_id=None*, *email=None*, *body=None*)
Create a MessageThread.

```
>>> message_thread = Intercom.create_message_thread(
... email="somebody@example.com",
... body="Hey Intercom, What is up?")
>>> message_thread['thread_id']
5591
>>> len(message_thread['messages'])
3
>>> message_thread['messages'][0]['html']
u'<p>Hey Intercom, What is up?</p>
```

'

classmethod create_note(*user_id=None*, *email=None*, *body=None*)
Create a note.

```
>>> result = Intercom.create_note(email="somebody@example.com",
... body="This is a note")
>>> result['html']
u'<p>This is a note</p>'
```

```
>>> result['user']['email']
u'somebody@example.com'
```

classmethod create_tag(*name*, *tag_or_untag*, *user_ids=None*, *emails=None*)

Create a tag (and maybe tag users).

```
>>> tag = Intercom.create_tag("Free Trial", "tag",
... user_ids=["abc123", "def456"])
>>> tag['id'] != None
True
>>> tag['name']
u'Free Trial'
>>> tag['tagged_user_count']
2
```

classmethod create_user(***kwargs*)

Creates a user.

N.B. Social and geo location data is fetched asynchronously, so a secondary call to users will be required to fetch it.

Arguments

- `user_id`: required (if no email) — a unique string identifier for the user
- `email`: required (if no user_id) — the user's email address
- `name`: The user's full name
- `created_at`: A UNIX timestamp representing the date the user was created
- `custom_data`: A hash of key/value pairs containing any other data about the user you want Intercom to store.
- `last_seen_ip`: An ip address (e.g. "1.2.3.4") representing the last ip address the user visited your application from. (Used for updating location_data)
- `last_seen_user_agent`: The user agent the user last visited your application with.
- `companies`: An array of hashes describing the companies this user belongs to. Currently companies are not returned in the response.
- `last_request_at or last_impression_at`: A UNIX timestamp representing the date the user last visited your application.
- `unsubscribed_from_emails`: A boolean value representing the users unsubscribed status.

```
>>> user = Intercom.create_user(user_id='7902',
... email='ben@intercom.io',
... name='Somebody', created_at=1270000000, last_seen_ip='1.2.3.4',
... custom_data={ 'app_name': 'Genesis'}, last_request_at=1300000000)
>>> user['name']
u'Somebody'
>>> user['custom_data']['app_name']
u'Genesis'
>>> user['last_impression_at']
1300000000
```

classmethod delete_user(*user_id=None*, *email=None*)

Delete a user.

```
>>> user = Intercom.get_user(user_id='123')
>>> user['email']
u'somebody@example.com'
```

classmethod get_message_threads (*user_id=None*, *email=None*, *thread_id=None*)

If a thread_id is specified, this returns a specific MessageThread (if it can find one), otherwise it returns all MessageThreads for the particular user.

```
>>> message_threads = Intercom.get_message_threads(
... email="somebody@example.com")
>>> type(message_threads)
<type 'list'>
>>> message_thread = Intercom.get_message_threads(
... email="somebody@example.com", thread_id=5591)
>>> type(message_thread)
<type 'dict'>
```

classmethod get_tag (*name=None*)

Return a dict for the tag by the specified name.

```
>>> tag = Intercom.get_tag(name="Free Trial")
>>> tag['id'] != None
True
>>> tag['name']
u'Free Trial'
>>> tag['tagged_user_count']
2
```

classmethod get_user (*email=None*, *user_id=None*)

Return a dict for the user represented by the specified email or user_id.

```
>>> user = Intercom.get_user(user_id='123')
>>> user['name']
u'Somebody'
```

classmethod get_users (***kwargs*)

Returns a paginated list of all users in your application on Intercom.

Arguments

- `page`: optional (defaults to 1)
- `per_page`: optional (defaults to 500, max value of 500)
- `tag_id`: optional — query for users that are tagged with a specific tag.
- `tag_name`: optional — query for users that are tagged with a specific tag.

Response

- `users`: an array of User objects (same as returned by getting a single User)
- `total_count`: the total number of Users tracked in your Intercom application
- `page`: the current requested page
- `next_page`: the next page number, if any
- `previous_page`: the previous page number, if any
- `total_pages`: the total number of pages

```
>>> result = Intercom.get_users()
>>> type(result)
<type 'dict'>
>>> len(result['users'])
3
```

classmethod reply_message_thread(*user_id=None*, *email=None*, *thread_id=None*, *body=None*, *read=None*)

Reply to the specific thread.

```
>>> message_thread = Intercom.reply_message_thread(
... email="somebody@example.com",
... thread_id=5591,
... body="If you're not talking to me you must be talking to someone")
>>> len(message_thread)
9
>>> message_thread['thread_id']
5591
>>> len(message_thread['messages'])
3
```

timeout = 10

classmethod update_tag(*name*, *tag_or_untag*, *user_ids=None*, *emails=None*)

Update a tag (and maybe tag users).

```
>>> tag = Intercom.update_tag("Free Trial", "tag",
... user_ids=["abc123", "def456"])
>>> tag['id'] != None
True
>>> tag['name']
u'Free Trial'
>>> tag['tagged_user_count']
2
```

classmethod update_user(***kwargs*)

Update a user with the available parameters.

```
>>> user = Intercom.get_user(user_id='123')
>>> user['name']
u'Somebody'
>>> user = Intercom.update_user(user_id='123', name='Guido')
>>> user['name']
u'Guido'
```

exception intercom.intercom.**IntercomError**(*message*, *result=None*)

Bases: exceptions.Exception

Base error.

exception intercom.intercom.**ResourceNotFound**(*message*, *result=None*)

Bases: *intercom.intercom.IntercomError*

Raised when a resource cannot be found e.g. a non-existant User.

exception intercom.intercom.**ServerError**(*message*, *result=None*)

Bases: *intercom.intercom.IntercomError*

Raised when the API returns an error other than an auth or not found.

exception intercom.intercom.**ServiceUnavailableError**(*message*, *result=None*)

Bases: *intercom.intercom.IntercomError*

Raised when the API cannot be handle a request.

intercom.intercom.**api_call**(*func_to_decorate*)
Decorator for handling AWS credentials.

intercom.intercom.**raise_errors_on_failure**(*response*)

2.1.5 message_thread Module

message_thread module.

```
>>> from intercom import Intercom
>>> Intercom.app_id = 'dummy-app-id'
>>> Intercom.api_key = 'dummy-api-key'
>>> from intercom import MessageThread
```

class intercom.message_thread.**Message**
Bases: dict

Object representing a Message in a MessageThread.

```
>>> message_thread = MessageThread.find(email="somebody@example.com",
... thread_id=5591)
>>> message = message_thread.messages[0]
>>> type(message.author)
<class 'intercom.message_thread.MessageAuthor'>
>>> message.html
u'<p>Hey Intercom, What is up?</p>
```

** '**

```
>>> type(message.created_at)
<type 'datetime.datetime'>
```

author
Returns who authored the message.

created_at
Returns a datetime for when this Message was created.

html
Returns the HTML body of the Message.

class intercom.message_thread.**MessageAuthor**
Bases: dict

Object represting the author of a Message.

```
>>> message_thread = MessageThread.find(email="somebody@example.com",
... thread_id=5591)
>>> author = message_thread.messages[0].author
>>> author.admin
False
>>> author.email
u'bob@example.com'
>>> author.user_id
u'456'
>>> author.avatar_path_50
```

```
>>> author.name
u'Bob'
```

admin
Returns whether the author is an admin.

avatar_path_50
Returns a URL to a 50x50 avatar of the author.

email
Returns the email address of the author.

name
Returns the author's name.

user_id
Returns the user_id of the author.

class intercom.message_thread.**MessageThread**
Bases: dict

An Intercom conversation between an admin and a User.

body

classmethod create(*user_id=None*, *email=None*, *body=None*)
Creates a new converstion.

```
>>> email = "somebody@example.com"
>>> body = "Hi everybody, I'm Doctor Nick"
>>> message_thread = MessageThread.create(email=email, body=body)
>>> message_thread.thread_id
5591
>>> len(message_thread.messages)
3
```

created_at
Sets a timestamp of when the last update occurred.

classmethod find(*user_id=None*, *email=None*, *thread_id=None*)
Finds a particular conversation for a particular user.

```
>>> message_thread = MessageThread.find(email="somebody@example.com")
Traceback (most recent call last):
    ...
ValueError: No thread_id specified
>>> message_thread = MessageThread.find(email="somebody@example.com",
... thread_id=5591)
>>> len(message_thread.messages)
3
>>> message = message_thread.messages[0]
>>> type(message)
<class 'intercom.message_thread.Message'>
```

classmethod find_all(*user_id=None*, *email=None*)
Finds all Messages for a particular user.

```
>>> message_threads = MessageThread.find_all(
... email="somebody@example.com")
>>> len(message_threads)
1
```

messages
Returns a list of Messages in this MessageThread.

read
Returns whether this thread has been read or not.

classmethod reply (*user_id=None*, *email=None*, *thread_id=None*, *body=None*, *read=None*)
Reply to an existing conversation.

```
>>> email = "somebody@example.com"
>>> thread_id = 5591
>>> body = "Are you talking to me?"
>>> message_thread = MessageThread.reply(email=email,
... thread_id=thread_id, body=body)
>>> len(message_thread.messages)
3
>>> message_thread.messages[0].html
u'<p>Hey Intercom, What is up?</p>
```

** '**

```
>>> message_thread.messages[1].html
u'<p>Not much, you?
```

** '**

```
>>> message_thread.messages[2].html
u'<p>Not much either :(</p>
```

'

set_body (*value*)
Set the body of a reply.

thread_id
Returns the thread_id of this MessageThread.

updated_at
Returns a datetime of when the last update occurred.

2.1.6 user Module

User module.

```
>>> from intercom import Intercom
>>> from intercom import User
```

class intercom.user.**Company**
Bases: dict

Object represents an Intercom Company

created_at
Returns the datetime this Company was created.

id
Returns the company's id.

name
Returns the company name e.g. Intercom.

class intercom.user.**CustomData**
Bases: dict

A dict that limits keys to strings, and values to real numbers and strings.

```
>>> from intercom.user import CustomData
>>> data = CustomData()
>>> data['a_dict'] = {}
Traceback (most recent call last):
    ...
ValueError: custom data only allows string and real number values
>>> data[1] = "a string"
Traceback (most recent call last):
    ...
ValueError: custom data only allows string keys
```

class intercom.user.**LocationData**
Bases: dict

Object representing a user's location data

This object is read-only, and to hint at this __setitem__ is disabled.

```
>>> from intercom.user import SocialProfile
>>> profile = SocialProfile(type=u'twitter')
>>> profile.type
u'twitter'
>>> profile['type'] = 'facebook'
Traceback (most recent call last):
    ...
NotImplementedError
```

city_name
The city name.

continent_code
The continent code.

country_code
The country code.

country_name
The country name.

latitude
Latitude.

longitude
Longitude.

postal_code
The postal code.

region_name
The region name.

timezone
The timezone.

class intercom.user.**SocialProfile**
Bases: dict

Object representing http://docs.intercom.io/#SocialProfiles)

This object is read-only, and to hint at this __setitem__ is disabled.

```
>>> from intercom.user import SocialProfile
>>> profile = SocialProfile(type=u'twitter')
>>> profile.type
u'twitter'
>>> profile['type'] = 'facebook'
Traceback (most recent call last):
    ...
NotImplementedError
```

id
The id

type
The type e.g. twitter, facebook, flickr, etc.

url
The profile url e.g. http://twitter.com/somebody

username
The profile username e.g. somebody

class intercom.user.**User**
Bases: *intercom.user.UserId*

Object representing http://docs.intercom.io/#UserData).

classmethod all()
Return all of the Users.

```
>>> users = User.all()
>>> len(users)
3
>>> users[0].email
u'first.user@example.com'
```

attributes = ('user_id', 'email', 'name', 'created_at', 'custom_data', 'last_seen_ip', 'last_seen_user_agent', 'companie

companies
Get the companies of a user. Currently unsupported by the Intercom API so an AttributeError is raised.

```
>>> user = User(email="somebody@example.com")
>>> user.companies
Traceback (most recent call last):
    ...
AttributeError: companies is a write-only property
```

company
Get the company of a user. Currently unsupported by the Intercom API so an AttributeError is raised.

```
>>> user = User(email="somebody@example.com")
>>> user.company
Traceback (most recent call last):
    ...
AttributeError: company is a write-only property
```

classmethod create(***kwargs*)
Create or update a user.

```
>>> user = User.create(email="somebody@example.com",last_impression_at=1400000000)
>>> user.name
u'Somebody'
>>> user.last_impression_at.year
2011
```

created_at
Returns the datetime this User started using your application.

custom_data
Returns a CustomData object for this User.

```
>>> users = User.all()
>>> custom_data = users[0].custom_data
>>> type(custom_data)
<class 'intercom.user.CustomData'>
>>> custom_data['monthly_spend']
155.5
```

classmethod delete(*user_id=None*, *email=None*)
Deletes a user.

```
>>> user = User.delete(email="somebody@example.com")
>>> user.user_id
u'123'
>>> user = User.delete(user_id="123")
>>> user.email
u'somebody@example.com'
```

classmethod find(*user_id=None*, *email=None*)
Find a user by email or user_id.

```
>>> user = User.find(email="somebody@example.com")
>>> user.user_id
u'123'
>>> user.name
u'Somebody'
>>> user = User.find(user_id=123)
>>> user.email
u'somebody@example.com'
>>> user.name
u'Somebody'
```

classmethod find_by_email(*email*)
Find a user by email.

```
>>> user = User.find_by_email("somebody@example.com")
>>> user.user_id
u'123'
>>> user.name
u'Somebody'
```

classmethod find_by_user_id(*user_id*)
Find a user by user_id.

```
>>> user = User.find(user_id=123)
>>> user.email
```

```
u'somebody@example.com'
>>> user.name
u'Somebody'
```

last_impression_at
Returns the datetime this User last used your application.

last_request_at
Get last time this User interacted with your application.

last_seen_ip
Returns the last seen IP address.

last_seen_user_agent
Returns the last seen User Agent.

location_data
Returns a LocationData object for this User.

```
>>> users = User.all()
>>> location_data = users[0].location_data
>>> type(location_data)
<class 'intercom.user.LocationData'>
>>> location_data.country_name
u'Chile'
>>> location_data.city_name
u'Santiago'
```

name
Returns the name e.g. Joe Bloggs.

relationship_score
Returns the relationship score.

save()
Creates or updates a User.

```
>>> user = User()
>>> user.email = "somebody@example.com"
>>> user.save()
>>> user.name
u'Somebody'
```

session_count
Returns how many sessions this User has used on your application.

social_profiles
Returns a list of SocialProfile objects for this User.

```
>>> users = User.all()
>>> social_profiles = users[0].social_profiles
>>> len(social_profiles)
2
>>> type(social_profiles[0])
<class 'intercom.user.SocialProfile'>
>>> social_profiles[0].type
u'twitter'
>>> social_profiles[0].url
u'http://twitter.com/abc'
```

unsubscribed_from_emails
Returns whether or not the user has unsubscribed from emails

class intercom.user.**UserId**
Bases: dict

Base class for objects that required user_id and email properties.

email
Returns the email address.

user_id
Returns the user_id.

CHAPTER 3

Changelog

- **0.2.13**
  - fix wildcard import from *intercom* **'#28 https://github.com/jkeyes/python-intercom/pull/28'_**. (https://github.com/marselester)
- **0.2.12**
  - include RTD theme in requirements.txt
- **0.2.11**
  - support for events. #25. (https://github.com/mekza)
  - using RTD theme for documentation.
  - updated links to Intercom API docs.
- **0.2.10**
  - basic support for companies. #18. (https://github.com/cameronmaske)
  - fixed User.delete. #19. (https://github.com/cameronmaske)
  - updated links to Intercom API docs.
  - Doctest fixes.
- **0.2.9**
  - *unsubscribed_from_emails* attribute added to *User* object. #15. (https://github.com/sdorazio)
  - *last_request_at* parameter supported in *Intercom.create_user*. #16.
  - page, per_page, tag_id, and tag_name parameters support on *Intercom.get_users*. #17.
- **0.2.8**
  - Added support for tagging #13.
  - Now installs into a clean python environment (https://github.com/vrachil) #12.
  - Doctest fix.
  - PEP8 formatting.
- **0.2.7**
  - Update delete user support to send bodyless request.
  - Add support for user notes.
- **0.2.6**

  - Support for delete user.

- **0.2.5**
  - Consistent version numbering (docs and code).
- **0.2.4**
  - Handle invalid JSON responses. (https://github.com/marselester)
  - Fix to doctests to pass with current Intercom dummy API.
- **0.2.3**
  - Fixed version number of distribution to match documentation.
- **0.2.2**
  - Finished docstrings and doctests.
- **0.2.1**
  - Added some docstrings.
- **0.2**
  - created source distribution #2.
  - renamed errors #1.
- **0.1**
  - initial release

CHAPTER 4

Development

4.1 Running the tests

Run all of the (nose) tests:

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

Run the unit tests:

```
nosetests tests -e integration
```

Run the integration tests (using the dummy *app_id* and *api_key*):

```
nosetests tests -e unit
```

4.2 Doctests

Run all of the doctests:

```
./bin/doctest
```

Run the doctests in a specific module:

```
./bin/doctest intercom/user.py
```

4.3 Code coverage

Generate a code coverage report:

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

4.4 Pylint

Generate a pylint report for a specific module:

```
pylint --rcfile=pylint.conf intercom/user.py
```

Generate a full pylint report:

```
pylint --rcfile=pylint.conf intercom
```

4.5 Runtime Dependencies

- Requests – an HTTP library "for human beings"

4.6 Development Dependencies

- nose – makes unit testing easier.
- coverage – code coverage.
- mock – patching methods for unit testing.
- pylint – source code analyzer.
- Sphinx – documentation decorator.
- Pygments – Python syntax highlighting for documentation.
- docutils – reStructuredText support.
- Jinja – templating language.

CHAPTER 5

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 6

Usage

6.1 Authentication

Intercom documentation: Authentication.

```
from intercom import Intercom
Intercom.app_id = 'dummy-app-id'
Intercom.api_key = 'dummy-api-key'
```

6.2 Users

6.2.1 Getting all Users

Intercom documentation: Getting all Users.

```
from intercom import User
for user in User.all():
    print user.email
```

6.2.2 Getting a User

Intercom documentation: Getting a User.

```
user = User.find(email="ben@intercom.io")
```

6.2.3 Create a User

Intercom documentation: Create a User.

```
user = User.create(email="ben@intercom.io",
        user_id=7902,
        name="Ben McRedmond",
        created_at=datetime.now(),
        custom_data={"plan": "pro"},
        last_seen_ip="1.2.3.4",
        last_seen_user_agent="ie6")
```

6.2.4 Updating a User

Intercom documentation: Updating a User.

```
user = User.find(email="ben@intercom.io")
user.name = "Benjamin McRedmond"
user.save()
```

6.2.5 Deleting a User

Intercom documentation: Deleting a User.

```
deleted_user = User.delete(email="ben@intercom.io")
```

6.3 Notes

6.3.1 Creating a Note

Intercom documentation: Creating a Note.

```
from intercom import Note
note = Note.create(email="ben@intercom.io",
        body="These are a few of my favourite things.")
```

6.4 Tagging

6.4.1 Getting a Tag

Intercom documentation: Getting a Tag.

```
from intercom import Tag
tag = Tag.find_by_name("Free Trial")
```

6.4.2 Creating a new Tag

Intercom documentation: Creating a new Tag.

```
from intercom import Tag
tag = Tag.create("Free Trial")
```

6.4.3 Updating an already existing Tag

Intercom documentation: Updating a Tag.

```
from intercom import Tag
tag = Tag.update("Free Trial", "tag",
    user_ids=["abc123", "def456"])
```

6.5 Impressions

6.5.1 Creating an Impression

Intercom documentation: Creating an Impression.

```
from intercom import Impression
impression = Impression.create(email="ben@intercom.io",
        user_agent="my-awesome-android-app-v0.0.1")
```

6.6 Message Threads

6.6.1 Getting Message Threads

Intercom documentation: Getting Message Threads.

```
from intercom import MessageThread

# all message threads
message_threads = MessageThread.find_all(email="ben@intercom.io")

# a specific thread
message_threads = MessageThread.find_all(email="ben@intercom.io",
        thread_id=123)
```

6.6.2 Creating a Message Thread

Intercom documentation: Creating a Message Thread.

```
message_thread = MessageThread.create(email="ben@intercom.io",
        body="Hey Intercom, What is up?")
```

6.6.3 Replying on a Message Thread

Intercom documentation: Replying on a Message Thread.

```
message_thread = MessageThread.create(email="ben@intercom.io",
        thread_id=123,
        body="Not much either :(")
```

6.7 Events

6.7.1 Submitting Events

Intercom documentation: Submitting Events.

```
from intercom import Event
impression = Event.create(event_name="sent-invite",
        user_id="314159")
```

CHAPTER 7

Development

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

CHAPTER 8

Changelog

The Changelog keeps track of changes per release.

CHAPTER 9

pydoc

View the extensive pydoc which has liberal helpings of *doctests* to display usage.

i

A

B

C

P

R

S

T

U