# python-slackclient Documentation

***Release 1.0.1***

**Ryan Huber, Jeff Ammons**

**Apr 03, 2018**

# Contents

CHAPTER 1

# Handling tokens and other sensitive data

Slack tokens are the keys to your—or your customers'—teams. Keep them secret. Keep them safe. One way to do that is to never explicitly hardcode them.

Try to avoid this when possible:

```
token = 'xoxb-abc-1232'
```

If you commit this code to GitHub, the world gains access to this token's team. Rather, we recommend you pass tokens in as environment variables, or persist them in a database that is accessed at runtime. You can add a token to the environment by starting your app as:

```
SLACK_BOT_TOKEN="xoxb-abc-1232" python myapp.py
```

Then in your code retrieve the key with:

```
import os
SLACK_BOT_TOKEN = os.environ["SLACK_BOT_TOKEN"]
```

You can use the same technique for other kinds of sensitive data that ne'er-do-wells could use in nefarious ways, including

- Incoming webhook URLs
- Slash command verification tokens
- App client ids and client secrets

For additional information, please see our Safely Storing Credentials page.

CHAPTER 2

# Test Tokens

During development (prior to implementing OAuth) you can use a test token provided by the Test Token Generator. These tokens provide access to your private data and that of your team.

**Tester tokens are not intended to replace OAuth 2.0 tokens.** Once your app is ready for users, replace this token with a proper OAuth token implementation.

**Never share test tokens with other users or applications. Do not publish test tokens in public code repositories.**

CHAPTER 3

# The OAuth flow

Authentication for Slack's APIs is done using OAuth, so you'll want to read up on OAuth.

In order to implement OAuth in your app, you will need to include a web server. In this example, we'll use Flask.

As mentioned above, we're setting the app tokens and other configs in environment variables and pulling them into global variables.

Depending on what actions your app will need to perform, you'll need different OAuth permission scopes. Review the available scopes here.

```
import os
from flask import Flask, request
from slackclient import SlackClient

client_id = os.environ["SLACK_CLIENT_ID"]
client_secret = os.environ["SLACK_CLIENT_SECRET"]
oauth_scope = os.environ["SLACK_BOT_SCOPE"]

app = Flask(__name__)
```

**The OAuth initiation link:**

To begin the OAuth flow, you'll need to provide the user with a link to Slack's OAuth page. This directs the user to Slack's OAuth acceptance page, where the user will review and accept or refuse the permissions your app is requesting as defined by the requested scope(s).

For the best user experience, use the Add to Slack button to direct users to approve your application for access or Sign in with Slack to log users in.

```
@app.route("/begin_auth", methods=["GET"])
def pre_install():
  return '''
      <a href="https://slack.com/oauth/authorize?scope={0}&client_id={1}">
          Add to Slack
      </a>
  '''.format(oauth_scope, client_id)
```

### The OAuth completion page

Once the user has agreed to the permissions you've requested on Slack's OAuth page, Slack will redirect the user to your auth completion page. Included in this redirect is a `code` query string param which you'll use to request access tokens from the `oauth.access` endpoint.

Generally, Web API requests require a valid OAuth token, but there are a few endpoints which do not require a token. `oauth.access` is one example of this. Since this is the endpoint you'll use to retrieve the tokens for later API requests, an empty string `""` is acceptable for this request.

```
@app.route("/finish_auth", methods=["GET", "POST"])
def post_install():

  # Retrieve the auth code from the request params
  auth_code = request.args['code']

  # An empty string is a valid token for this request
  sc = SlackClient("")

  # Request the auth tokens from Slack
  auth_response = sc.api_call(
    "oauth.access",
    client_id=client_id,
    client_secret=client_secret,
    code=auth_code
  )
```

A successful request to `oauth.access` will yield two tokens: A `user` token and a `bot` token. The `user` token `auth_response['access_token']` is used to make requests on behalf of the authorizing user and the `bot` token `auth_response['bot']['bot_access_token']` is for making requests on behalf of your app's bot user.

If your Slack app includes a bot user, upon approval the JSON response will contain an additional node containing an access token to be specifically used for your bot user, within the context of the approving team.

When you use Web API methods on behalf of your bot user, you should use this bot user access token instead of the top-level access token granted to your application.

```
# Save the bot token to an environmental variable or to your data store
# for later use
os.environ["SLACK_USER_TOKEN"] = auth_response['access_token']
os.environ["SLACK_BOT_TOKEN"] = auth_response['bot']['bot_access_token']

# Don't forget to let the user know that auth has succeeded!
return "Auth complete!"
```

Once your user has completed the OAuth flow, you'll be able to use the provided tokens to make a variety of Web API calls on behalf of the user and your app's bot user.

See the *Web API usage* section of this documentation for usage examples.

CHAPTER 4

# Basic Usage

The Slack Web API allows you to build applications that interact with Slack in more complex ways than the integrations we provide out of the box.

This package is a modular wrapper designed to make Slack Web API calls simpler and easier for your app. Provided below are examples of how to interact with commonly used API endpoints, but this is by no means a complete list. Review the full list of available methods here.

See *Tokens & Authentication* for API token handling best practices.

## 4.1 Sending a message

The primary use of Slack is sending messages. Whether you're sending a message to a user or to a channel, this method handles both.

To send a message to a channel, use the channel's ID. For IMs, use the user's ID.

```
from slackclient import SlackClient

slack_token = os.environ["SLACK_API_TOKEN"]
sc = SlackClient(slack_token)

sc.api_call(
  "chat.postMessage",
  channel="C0XXXXXX",
  text="Hello from Python! :tada:"
)
```

There are some unique options specific to sending IMs, so be sure to read the **channels** section of the chat.postMessage page for a full list of formatting and authorship options.

Sending an ephemeral message, which is only visible to an assigned user in a specified channel, is nearly the same as sending a regular message, but with an additional `user` parameter.

```
from slackclient import SlackClient

slack_token = os.environ["SLACK_API_TOKEN"]
sc = SlackClient(slack_token)

sc.api_call(
  "chat.postEphemeral",
  channel="C0XXXXXX",
  text="Hello from Python! :tada:",
  user="U0XXXXXXX"
)
```

See chat.postEphemeral for more info.

---

## 4.2 Replying to messages and creating threads

Threaded messages are just like regular messages, except thread replies are grouped together to provide greater context to the user. You can reply to a thread or start a new threaded conversation by simply passing the original message's `ts` ID in the `thread_ts` attribute when posting a message. If you're replying to a threaded message, you'll pass the *thread_ts* ID of the message you're replying to.

A channel or DM conversation is a nearly linear timeline of messages exchanged between people, bots, and apps. When one of these messages is replied to, it becomes the parent of a thread. By default, threaded replies do not appear directly in the channel, instead relegated to a kind of forked timeline descending from the parent message.

```
from slackclient import SlackClient

slack_token = os.environ["SLACK_API_TOKEN"]
sc = SlackClient(slack_token)

sc.api_call(
  "chat.postMessage",
  channel="C0XXXXXX",
  text="Hello from Python! :tada:",
  thread_ts="1476746830.000003"
)
```

By default, `reply_broadcast` is set to `False`. To indicate your reply is germane to all members of a channel, set the `reply_broadcast` boolean parameter to `True`.

```
from slackclient import SlackClient

slack_token = os.environ["SLACK_API_TOKEN"]
sc = SlackClient(slack_token)

sc.api_call(
  "chat.postMessage",
  channel="C0XXXXXX",
  text="Hello from Python! :tada:",
  thread_ts="1476746830.000003",
  reply_broadcast=True
)
```

**Note:** While threaded messages may contain attachments and message buttons, when your reply is broadcast to the channel, it'll actually be a reference to your reply, not the reply itself. So, when appearing in the channel, it won't contain any attachments or message buttons. Also note that updates and deletion of threaded replies works the same as regular messages.

See the Threading messages together article for more information.

---

## 4.3 Updating the content of a message

Let's say you have a bot which posts the status of a request. When that request is updated, you'll want to update the message to reflect it's state. Or your user might want to fix a typo or change some wording. This is how you'll make those changes.

```
from slackclient import SlackClient

slack_token = os.environ["SLACK_API_TOKEN"]
sc = SlackClient(slack_token)

sc.api_call(
  "chat.update",
  ts="1476746830.000003",
  channel="C0XXXXXX",
  text="Hello from Python! :tada:"
)
```

See chat.update for formatting options and some special considerations when calling this with a bot user.

---

## 4.4 Deleting a message

Sometimes you need to delete things.

```
from slackclient import SlackClient

slack_token = os.environ["SLACK_API_TOKEN"]
sc = SlackClient(slack_token)

sc.api_call(
  "chat.delete",
  channel="C0XXXXXX",
  ts="1476745373.000002"
)
```

See chat.delete for more info.

---

## 4.5 Adding or removing an emoji reaction

You can quickly respond to any message on Slack with an emoji reaction. Reactions can be used for any purpose: voting, checking off to-do items, showing excitement — and just for fun.

This method adds a reaction (emoji) to an item (`file`, `file comment`, `channel message`, `group message`, or `direct message`). One of file, file_comment, or the combination of channel and timestamp must be specified.

```
from slackclient import SlackClient

slack_token = os.environ["SLACK_API_TOKEN"]
sc = SlackClient(slack_token)

sc.api_call(
  "reactions.add",
  channel="C0XXXXXXX",
  name="thumbsup",
  timestamp="1234567890.123456"
)
```

Removing an emoji reaction is basically the same format, but you'll use `reactions.remove` instead of `reactions.add`

```
sc.api_call(
  "reactions.remove",
  channel="C0XXXXXXX",
  name="thumbsup",
  timestamp="1234567890.123456"
)
```

See reactions.add and reactions.remove for more info.

---

## 4.6 Getting a list of channels

At some point, you'll want to find out what channels are available to your app. This is how you get that list.

**Note:** This call requires the `channels:read` scope.

```
from slackclient import SlackClient

slack_token = os.environ["SLACK_API_TOKEN"]
sc = SlackClient(slack_token)

sc.api_call("channels.list")
```

Archived channels are included by default. You can exclude them by passing `exclude_archived=1` to your request.

```
from slackclient import SlackClient

slack_token = os.environ["SLACK_API_TOKEN"]
sc = SlackClient(slack_token)

sc.api_call(
```

```
  "channels.list",
  exclude_archived=1
)
```

See channels.list for more info.

---

## 4.7 Getting a channel's info

Once you have the ID for a specific channel, you can fetch information about that channel.

```
from slackclient import SlackClient

slack_token = os.environ["SLACK_API_TOKEN"]
sc = SlackClient(slack_token)

sc.api_call(
  "channels.info",
  channel="C0XXXXXXX"
)
```

See channels.info for more info.

---

## 4.8 Joining a channel

Channels are the social hub of most Slack teams. Here's how you hop into one:

```
from slackclient import SlackClient

slack_token = os.environ["SLACK_API_TOKEN"]
sc = SlackClient(slack_token)

sc.api_call(
  "channels.join",
  channel="C0XXXXXXY"
)
```

If you are already in the channel, the response is slightly different. `already_in_channel` will be true, and a limited `channel` object will be returned. Bot users cannot join a channel on their own, they need to be invited by another user.

See channels.join for more info.

---

## 4.9 Leaving a channel

Maybe you've finished up all the business you had in a channel, or maybe you joined one by accident. This is how you leave a channel.

```
from slackclient import SlackClient

slack_token = os.environ["SLACK_API_TOKEN"]
sc = SlackClient(slack_token)

sc.api_call(
  "channels.leave",
  channel="C0XXXXXXX"
)
```

See channels.leave for more info.

---

## 4.10 Get a list of team members

```
from slackclient import SlackClient

slack_token = os.environ["SLACK_API_TOKEN"]
sc = SlackClient(slack_token)

sc.api_call("users.list")
```

See users.list for more info.

---

## 4.11 Uploading files

```
from slackclient import SlackClient

slack_token = os.environ["SLACK_API_TOKEN"]
sc = SlackClient(slack_token)

with open('thinking_very_much.png') as file_content:
    sc.api_call(
        "files.upload",
        channels="C3UKJTQAC",
        file=file_content,
        title="Test upload"
    )
```

See users.list for more info.

---

## 4.12 Web API Rate Limits

Slack allows applications to send no more than one message per second. We allow bursts over that limit for short periods. However, if your app continues to exceed the limit over a longer period of time it will be rate limited.

Here's a very basic example of how one might deal with rate limited requests.

If you go over these limits, Slack will start returning a HTTP 429 Too Many Requests error, a JSON object containing the number of calls you have been making, and a Retry-After header containing the number of seconds until you can retry.

```
from slackclient import SlackClient
import time

slack_token = os.environ["SLACK_API_TOKEN"]
sc = SlackClient(slack_token)

# Simple wrapper for sending a Slack message
def send_slack_message(channel, message):
  return sc.api_call(
    "chat.postMessage",
    channel=channel,
    text=message
  )

# Make the API call and save results to `response`
response = send_slack_message("C0XXXXXX", "Hello, from Python!")

# Check to see if the message sent successfully.
# If the message succeeded, `response["ok"]`` will be `True`
if response["ok"]:
  print("Message posted successfully: " + response["message"]["ts"])
  # If the message failed, check for rate limit headers in the response
elif response["ok"] is False and response["headers"]["Retry-After"]:
  # The `Retry-After` header will tell you how long to wait before retrying
  delay = int(response["headers"]["Retry-After"])
  print("Rate limited. Retrying in " + str(delay) + " seconds")
  time.sleep(delay)
  send_slack_message(message, channel)
```

See the documentation on Rate Limiting for more info.

# CHAPTER 5

# Conversations API

The Slack Conversations API provides your app with a unified interface to work with all the channel-like things encountered in Slack; public channels, private channels, direct messages, group direct messages, and our newest channel type, Shared Channels.

See Conversations API docs for more info.

## 5.1 Creating a direct message or multi-person direct message

This Conversations API method opens a multi-person direct message or just a 1:1 direct message.

*Use conversations.create for public or private channels.*

Provide 1 to 8 user IDs in the `user` parameter to open or resume a conversation. Providing only 1 ID will create a direct message. Providing more will create an `mpim`.

If there are no conversations already in progress including that exact set of members, a new multi-person direct message conversation begins.

Subsequent calls to `conversations.open` with the same set of users will return the already existing conversation.

```
from slackclient import SlackClient

slack_token = os.environ["SLACK_API_TOKEN"]
sc = SlackClient(slack_token)

sc.api_call(
  "conversations.open",
  users=["W1234567890","U2345678901","U3456789012"]
)
```

See conversations.open additional info.

## 5.2 Creating a public or private channel

Initiates a public or private channel-based conversation

```
from slackclient import SlackClient

slack_token = os.environ["SLACK_API_TOKEN"]
sc = SlackClient(slack_token)

sc.api_call(
  "conversations.create",
  name="myprivatechannel",
  is_private=True
)
```

See conversations.create additional info.

---

## 5.3 Getting information about a conversation

This Conversations API method returns information about a workspace conversation.

```
from slackclient import SlackClient

slack_token = os.environ["SLACK_API_TOKEN"]
sc = SlackClient(slack_token)

sc.api_call(
  "conversations.info",
  channel="C0XXXXXX",
)
```

See conversations.info for more info.

---

## 5.4 Getting a list of conversations

This Conversations API method returns a list of all channel-like conversations in a workspace. The "channels" returned depend on what the calling token has access to and the directives placed in the `types` parameter.

```
from slackclient import SlackClient

slack_token = os.environ["SLACK_API_TOKEN"]
sc = SlackClient(slack_token)

sc.api_call("conversations.list")
```

Only public conversations are included by default. You may include additional conversations types by passing `types` (as a string) into your list request. Additional conversation types include `public_channel` and `private_channel`.

```
from slackclient import SlackClient

slack_token = os.environ["SLACK_API_TOKEN"]
sc = SlackClient(slack_token)

# Note that `types` is a string
sc.api_call(
  "conversations.list",
  types="public_channel, private_channel"
)
```

See conversations.list for more info.

---

## 5.5 Leaving a conversation

Maybe you've finished up all the business you had in a conversation, or maybe you joined one by accident. This is how you leave a conversation.

```
from slackclient import SlackClient

slack_token = os.environ["SLACK_API_TOKEN"]
sc = SlackClient(slack_token)

sc.api_call(
  "conversations.leave",
  channel="C0XXXXXXX"
)
```

See conversations.leave for more info.

---

## 5.6 Get conversation members

Get a list fo the members of a conversation

```
from slackclient import SlackClient

slack_token = os.environ["SLACK_API_TOKEN"]
sc = SlackClient(slack_token)

sc.api_call("conversations.members",
  channel="C0XXXXXXX"
)
```

See users.list for more info.

CHAPTER 6

# Real Time Messaging

The Real Time Messaging API is a WebSocket-based API that allows you to receive events from Slack in real time and send messages as users.

If you prefer events to be pushed to you instead, we recommend using the HTTP-based Events API instead. Most event types supported by the RTM API are also available in the Events API.

See *Tokens & Authentication* for API token handling best practices.

## 6.1 Connecting to the Real Time Messaging API

```
from slackclient import SlackClient

slack_token = os.environ["SLACK_API_TOKEN"]
sc = SlackClient(slack_token)

if sc.rtm_connect():
  while sc.server.connected is True:
        print sc.rtm_read()
        time.sleep(1)
else:
    print "Connection Failed"
```

If you connect successfully the first event received will be a hello:

```
{
  u'type': u'hello'
}
```

If there was a problem connecting an error will be returned, including a descriptive error message:

```
{
  u'type': u'error',
    u'error': {
```

```
    u'code': 1,
    u'msg': u'Socket URL has expired'
  }
}
```

## 6.2 rtm.start vs rtm.connect

If you expect your app to be used on large teams, we recommend starting the RTM client with *rtm.connect* rather than the default connection method for this client, *rtm.start*. *rtm.connect* provides a lighter initial connection payload, without the team's channel and user information included. You'll need to request channel and user info via the Web API separately.

To do this, simply pass *with_team_state=False* into the *rtm_connect* call, like so:

```
from slackclient import SlackClient

slack_token = os.environ["SLACK_API_TOKEN"]
sc = SlackClient(slack_token)

if sc.rtm_connect(with_team_state=False):
    while True:
        print sc.rtm_read()
        time.sleep(1)
else:
    print "Connection Failed"
```

Passing *auto_reconnect=True* will tell the websocket client to automatically reconnect if the connection gets dropped.

See the rtm.start docs and the rtm.connect docs for more details.

## 6.3 RTM Events

```
{
  u'type': u'message',
  u'ts': u'1358878749.000002',
  u'user': u'U023BECGF',
  u'text': u'Hello'
}
```

See RTM Events for a complete list of events.

## 6.4 Sending messages via the RTM API

You can send a message to Slack by sending JSON over the websocket connection.

```
from slackclient import SlackClient

slack_token = os.environ["SLACK_API_TOKEN"]
sc = SlackClient(slack_token)

sc.rtm_send_message("welcome-test", "test")
```

You can send a message to a private group or direct message channel in the same way, but using a Group ID (`C024BE91L`) or DM channel ID (`D024BE91L`).

You can send a message in reply to a thread using the `thread` argument, and optionally broadcast that message back to the channel by setting `reply_broadcast` to `True`.

```
from slackclient import SlackClient

slack_token = os.environ["SLACK_API_TOKEN"]
sc = SlackClient(slack_token)

sc.rtm_send_message("welcome-test", "test", "1482960137.003543", True)
```

See Threading messages for more details on using threads.

The RTM API only supports posting messages with basic formatting. It does not support attachments or other message formatting modes.

To post a more complex message as a user, see *Web API usage*.

CHAPTER 7

# Frequently Asked Questions

## 7.1 What even is Slack Developer Kit for Python and why should I care?

Slack Developer Kit for Python is a wrapper around commonly accessed parts of the Slack Platform. It provides basic mechanisms for using the Slack Web API from within your Python app.

On the other hand, Slack Developer Kit for Python does not provide access to the RTM or Events bot-building APIs, although we anticipate adding support for these in the future.

## 7.2 OMG I found a bug!

Well, poop. Take a deep breath, and then let us know on the Issue Tracker. If you're feeling particularly ambitious, why not submit a pull request with a bug fix?

## 7.3 Hey, there's a feature missing!

There's always something more that could be added! You can let us know in the Issue Tracker to start a discussion around the proposed feature, that's a good start. If you're feeling particularly ambitious, why not write the feature yourself, and submit a pull request! We love feedback and we love help and we don't bite. Much.

## 7.4 I'd like to contribute. . . but how?

What an excellent question. First of all, please have a look at our general contributing guidelines. We'll wait for you here.

All done? Great! While we're super excited to incorporate your new feature into Slack Developer Kit for Python, there are a couple of things we want to make sure you've given thought to.

- Please write unit tests for your new code. But don't **just** aim to increase the test coverage, rather, we expect you to have written **thoughtful** tests that ensure your new feature will continue to work as expected, and to help future contributors to ensure they don't break it!
- Please document your new feature. Think about **concrete use cases** for your feature, and add a section to the appropriate document, including a **complete** sample program that demonstrates your feature. Don't forget to update the changelog in `changelog.rst`!

Including these two items with your pull request will totally make our day—and, more importantly, your future users' days!

On that note…

## 7.5 How do I compile the documentation?

This project's documentation is generated with Sphinx. If you are editing one of the many reStructuredText files in the `docs-src` folder, you'll need to rebuild the documentation. It is recommended to run the following steps inside a `virtualenv` environment.

```
tox -e docs
```

Do be sure to add the `docs` folder and its contents to your pull request!

CHAPTER 8

# Changelog

## 8.1 v1.2.1 (2018-03-26)

- Added rate limit handling for rtm connections (thanks @jayalane!)

## 8.2 v1.2.0 (2018-03-20)

- You can now tell the RTM client to automatically reconnect by passing *auto_reconnect=True*

## 8.3 v1.1.3 (2018-03-01)

- Fixed another API param encoding bug. It encodes things properly now.

## 8.4 v1.1.2 (2018-01-31)

- Fixed an encoding issue which was encoding some Web API params incorrectly (sorry)

## 8.5 v1.1.1 (2018-01-30)

- Adds HTTP response headers to *api_call* responses to expose things like rate limit info
- Moves *token* into auth header rather than request params

## 8.6 v1.1.0 (2017-11-21)

- Aadds new SlackClientError and ResponseParseError types to describe errors - thanks @aoberoi!
- Fix Build Error (#245) - thanks @stasfilin!
- include email as user property (#173) - thanks @acaire!
- Add http reply into slack login and slack connection error (#216) - thanks @harlowja!
- Removed unused exception class (#233)
- Fix rtm_send_message bug (#225) - thanks @kt5356!
- Allow use of custom parameters on rtm_connect() (#210) - thanks @kamushadenes!
- Fix link to rtm.connect docs (#223) - @sampart!

## 8.7 v1.0.9 (2017-08-31)

- Fixed rtm_send_message ID bug introduced in 1.0.8

## 8.8 v1.0.8 (2017-08-31)

- Added rtm.connect support

## 8.9 v1.0.7 (2017-08-02)

- Fixes an issue where connecting over RTM to large teams may result in “Websocket URL expired” errors
- A load of packaging improvements

## 8.10 v1.0.6 (2017-06-12)

- Added proxy support (thanks @timfeirg!)
- Tidied up docs (thanks @schlueter!)
- Added tox settings for Python 3 testing (thanks @cclauss!)

## 8.11 v1.0.5 (2017-01-23)

- Allow RTM Channel.send_message to reply to a thread
- Index users by ID instead of Name (non-breaking change)
- Added timeout to api calls.
- Fixed a typo about token access in auth.rst, thanks @kelvintaywl!
- Added Message Threads to the docs

## 8.12 v1.0.4 (2016-12-15)

- fixed the ability to search for a user by ID

## 8.13 v1.0.3 (2016-12-13)

- fixed an issue causing RTM connections to fail for large teams

## 8.14 v1.0.2 (2016-09-22)

- removed unused ping counter
- fixed contributor guidelines links
- updated documentation
- Fix bug preventing API calls requiring a file ID
- Removes files from api_calls before JSON encoding, so the request is properly formatted

## 8.15 v1.0.1 (2016-03-25)

- fix for __eq__ comparison in channels using '#' in channel name
- added copyright info to the LICENSE file

## 8.16 v1.0.0 (2016-02-28)

- the `api_call` function now returns a decoded JSON object, rather than a JSON encoded string
- some `api_call` calls now call actions on the parent server object: - `im.open` - `mpim.open`, `groups.create`, `groups.createChild` - `channels.create`, *channels.join*'

## 8.17 v0.18.0 (2016-02-21)

- Moves to use semver for versioning
- Adds support for private groups and MPDMs
- Switches to use requests instead of urllib
- Gets Travis CI integration working
- Fixes some formatting issues so the code will work for python 2.6
- Cleans up some unused imports, some PEP-8 fixes and a couple bad default args fixes

## 8.18 v0.17.0 (2016-02-15)

- Fixes the server so that it doesn't add duplicate users or channels to its internal lists, https://github.com/slackapi/python-slackclient/commit/0cb4bcd6e887b428e27e8059b6278b86ee661aaa
- README updates: - Updates the URLs pointing to Slack docs for configuring authentication, https://github.com/slackapi/python-slackclient/commit/7d01515cebc80918a29100b0e4793790eb83e7b9 - s/channnels/channels, https://github.com/slackapi/python-slackclient/commit/d45285d2f1025899dcd65e259624ee73771f94bb
- Adds users to the local cache when they join the team, https://github.com/slackapi/python-slackclient/commit/f7bb8889580cc34471ba1ddc05afc34d1a5efa23
- Fixes urllib py 2/3 compatibility, https://github.com/slackapi/python-slackclient/commit/1046cc2375a85a22e94573e2aad954ba7287c886

CHAPTER 9

# About

## 9.1 Slack Developer Kit for Python

Access the Slack Platform from your Python app. Slack Developer Kit for Python lets you build on the Slack Web APIs pythonically.

Slack Developer Kit for Python is proudly maintained with  by the Slack Developer Tools team

- License
- Code of Conduct
- Contributing
- Contributor License Agreement

CHAPTER 10

# Slack Developer Kit for Python

Whether you're building a custom app for your team, or integrating a third party service into your Slack workflows, Slack Developer Kit for Python allows you to leverage the flexibility of Python to get your project up and running as quickly as possible.

## 10.1 Requirements and Installation

We recommend using PyPI to install Slack Developer Kit for Python

```
pip install slackclient
```

Of course, if you prefer doing things the hard way, you can always implement Slack Developer Kit for Python by pulling down the source code directly into your project:

```
git clone https://github.com/slackapi/python-slackclient.git
pip install -r requirements.txt
```

## 10.2 Getting Help

If you get stuck, we're here to help. The following are the best ways to get assistance working through your issue:

- Use our Github Issue Tracker for reporting bugs or requesting features.
- Visit the Bot Developer Hangout for getting help using Slack Developer Kit for Python or just generally bond with your fellow Slack developers.