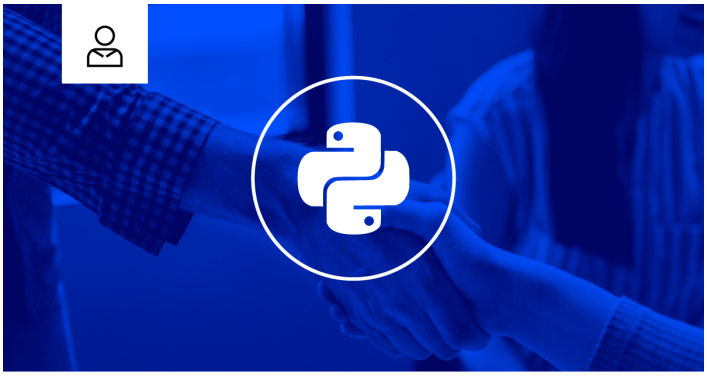

Sync Google and Microsoft Contacts in Python

Learn how to add contacts CRUD to your app with just a few lines of code.

Nylas | February 20, 2020

If you're building an application that enables your users to better communicate with candidates, prospects, or business partners, embedding their contacts data into your app can drastically improve the quality of connection and collaboration.

This can show up in myriad ways - having the most up-to-date email address, telephone number, and job title help keep data clean and accurate automatically, without requiring users to update this data themselves.

In this blog post, we'll take a look at how to view, create and delete contacts with the Nylas Python SDK, which connects your app to 100% of contact providers, including Gmail, Outlook, Office 365 and Yahoo.

Assuming you have pip installed on your machine and have created a virtual environment to install Nylas, you'll want to sign up for your Nylas developer account, and follow our guide to get your API keys and authorize your first email account. Once you've done that, you will have three tokens that you need to run the code examples below:

- CLIENT_ID - The CLIENT ID found on the dashboard page for your Nylas App
- CLIENT_SECRET - The CLIENT SECRET found on the dashboard page for your Nylas App
- ACCESS_TOKEN - The Access token provided when you authenticate an account to your Nylas App

Great! Go ahead and activate your virtual environment and run `pip install nylas`.

Creating a Contact with the Python SDK

Next, as we always do when using the Python SDK, import the APIClient class from the nylas package, and create a new instance of this class, passing in the tokens you obtained earlier.

```
from nylas import APIClient

nylas = APIClient(

    CLIENT_ID,

    CLIENT_SECRET,

    ACCESS_TOKEN

)
```

Now, let's say one of your app users, Francisco has made a new connection and wants to add him as a contact within your application.

We'll use the .create() method to create a new contact object for him.

```
contact = nylas.contacts.create()
```

He'll want to add a few details about this contact.

```
contact.given_name = "Napoleoooooone"

contact.middle_name = "di"

contact.surname = "Buonaparte"

contact.emails =

contact.notes = "Follow up with him next month!"

contact.phone_numbers =
```

```
contact.web_pages =
```

The contact object has many more attributes available to you, like office_location, job_title, company_name, and more, all of which you can view in our API reference.

Last but not least, we must remember to save the contact object we just created. The .create() and .save() methods together are the equivalent of making a call to our POST /contacts endpoint.

```
contact.save()
```

Saving the contact ensures that this contact will be saved to Nylas, which then syncs the new contact to the third-party provider (e.g. Gmail), so the next time your user logs into the email client, the newly created contact will be available to him without any additional configuring.

Updating a Contact with the Python SDK

But wait! Francisco realizes, he misspelled Napoleone's name and wants to correct it. Not a problem. To facilitate this, simply retrieve the contact that was created by passing in the contact ID to the .get() method, and update the attribute in question, and save the contact.

```
contact = nylas.contacts.get("{id}")

contact.given_name = "Napoleone"

contact.save()
```

Viewing all Contacts with the Python SDK

There may come a time when Francisco wants to review his contacts and prune any superfluous ones.

We can help him there. Simply fetch all contacts by calling the .all() method:

```
contacts = nylas.contacts.all()
```

This will return a list of all contact objects - if you'd like to winnow it down further to return the information to Francisco in a certain format, the data can be manipulated in a number of ways. For

example, if you wanted to return the contacts first and last name, email address, along with the ID of the contact object, you could do something like this:

```
for contact in contacts:

    email = list(contact.emails.values())

    print("Name: {} {} | Email: {} | ID: {}".format(

        contact.given_name, contact.surname, email, contact.id)

          )
```

Deleting a Contact with the Python SDK

In reviewing his contacts, Francisco remembers he heard that Napoleone moved to a deserted island without internet and that he'd better just go ahead and delete him. To aid Francisco's request, get the ID of the contact object he'd like to delete and pass it as an argument to the .delete() method, like so:

```
contact = nylas.contacts.get("{id}”)

nylas.contacts.delete(contact.id)
```

And just like that, your app has CRUD functionality for contacts. If you'd like a more detailed step-by-step guide on the code we've covered here, take a look at our tutorial. But that's not all! Further exploration of the Nylas API awaits.

Explore the Nylas API

Get Started - Build your first integration with Nylas in 15 minutes.

Quickstart Guides - Get up to speed quickly with our SDKs for Python, Node.js, and Ruby or explore the Nylas Email, Calendar, and Contacts APIs.

How Nylas Works - Take a look at the Nylas architecture to see how we sync billions of emails.

Tutorials - Check out our tutorials to learn how to carry out common functionality like creating,

reading and RSVP-ing to calendar events.

Integration Guides - Our integration guides cover what it takes to incorporate email functionality into your app. They cover best practices for using the Nylas Communications Platform and provider-specific advice for Google, Microsoft, IMAP, and more.

Set up Postman - Postman makes it easy to explore the Nylas Email API.