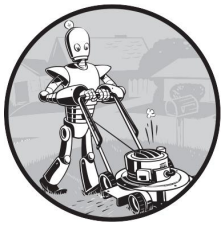
20

SENDING EMAIL, TEXTS, AND PUSH NOTIFICATIONS



Checking and replying to email is a huge time sink, and you can't just write a program to handle all your email for you, as each message requires its own response. But you can still automate plenty of email-related tasks once you know how to write programs that can send and receive email.

For example, maybe you have a spreadsheet full of customer records and want to send each customer a different form letter depending on their age and location details. Commercial software might not be able to do this for you. Fortunately, you can write your own program to send these emails, saving yourself a lot of time spent copying and pasting.

You can also write programs to send SMS text messages and push notifications to notify you of things even while you're away from your computer. If you're automating a task that takes a couple of hours to do, you probably don't want to go back to your computer every few minutes to check on the program's status. Instead, the program can just text your phone when it's done, freeing you to focus on more important things while you're away from your computer.

This chapter features the EZGmail module, a simple way to send and read emails from Gmail accounts, as well as the free *ntfy* service that provides push notifications between devices.

WARNING

I highly recommend that you set up a separate email account for any scripts that send or receive emails. This will prevent bugs in your programs from affecting your personal email account (by deleting emails or accidentally spamming your contacts, for example). It's a good idea to first do a dry run by commenting out the code that actually

sends or deletes emails and replacing it with a temporary `print()` *call. This way, you can test your program before running it for real.*

The Gmail API

Gmail owns close to one-third of the email client market share, and most likely you have at least one Gmail email address. Because of Gmail's additional security and anti-spam measures, controlling a Gmail account is better done through the EZGmail module than through the `smtplib` and `imaplib` modules in Python's standard library. I wrote EZGmail to work on top of the official Gmail API and provide functions that make it easy to use Gmail from Python. For full instructions on installing EZGmail, see Appendix A.

Enabling the API

Before you write code, you must first sign up for a Gmail email account at *https://gmail.com*. Then, you must set up the Gmail API for your account through the Google Cloud console at *https://console.cloud.google.com*. These steps are identical to the steps for setting up EZSheets detailed in Chapter 15, so I won't repeat them in this chapter. Create a new project and make sure to enable the Gmail API instead of the Google Sheets API. On step 2 of the OAuth consent screen configuration, add the *https://mail.google.com* scope to let your Python scripts read and send email. When you're done, you should have a credentials file and a token file.

Then, in the interactive shell, enter the following code:

```
>>> import ezgmail
>>> ezgmail.init()
```

If no error appears, EZGmail has been correctly installed.

Sending Mail

Once EZGmail is configured, you should be able to send email with a single function call:

```
>>> import ezgmail
>>> ezgmail.send('recipient@example.com',
'Subject line', 'Body of the email')
```

If you want to attach files to your email, you can provide an extra list argument to the `send()` function:

```
>>> ezgmail.send('recipient@example.com',
'Subject line', 'Body of the email',
['attachment1.jpg', 'attachment2.mp3'])
```

Note that as part of its security and anti-spam features, Gmail might not send repeated emails with the exact same text (as these are likely spam) or emails that contain *.exe* or *.zip* file attachments (as they are potentially viruses).

You can also supply the optional keyword arguments `cc` and `bcc` to send carbon copies and blind carbon copies:

```
>>> import ezgmail
>>> ezgmail.send('recipient@example.com',
'Subject line', 'Body of the
email', cc='friend@example.com',
bcc='otherfriend@example.com,
someoneelse@example.com')
```

If you need to remember which Gmail address the *token.json* file is configured for, you can examine `ezgmail.EMAIL_ADDRESS`:

```
>>> import ezgmail
>>> ezgmail.EMAIL_ADDRESS
'example@gmail.com'
```

Be sure to treat the *token.json* file in the same way as your password. If someone else obtains this file, they can access your Gmail account (though they won't be able to change your Gmail password). To revoke previously issued *token.json* files, return to the Google Cloud console and delete the credential for the compromised token. You will need to repeat the setup steps to generate a new credential and token file before you can resume using EZGmail.

Reading Mail

Gmail organizes email messages that are replies to each other into conversation threads. When you log in to Gmail in your web browser or through an app, you're really looking at email threads rather than individual emails (even if the thread has only one email in it).

EZGmail has `GmailThread` and `GmailMessage` objects to represent conversation threads and individual emails, respectively. A `GmailThread` object has a `messages` attribute that holds a list of `GmailMessage` objects. The `unread()` function returns a list of `GmailThread` objects for the 25 most recent unread emails, which can then be passed to `ezgmail.summary()` to print a summary of the conversation threads in that list:

```
>>> import ezgmail
>>> unread_threads = ezgmail.unread()
# List of GmailThread objects
>>> ezgmail.summary(unread_threads)
Al, Jon - Do you want to watch RoboCop this
weekend? - Dec 09
Jon - Thanks for stopping me from buying
Bitcoin. - Dec 09
```

The `summary()` function is handy for displaying a quick summary of the email threads, but to access specific messages (and parts of messages), you'll want to examine the `messages` attribute of the `GmailThread` object. The `messages` attribute contains a list of the `GmailMessage` objects that make up the thread, and these have `subject`, `body`, `timestamp`, `sender`, and `recipient` attributes that describe the email:

```
>>> len(unread_threads)
2
>>> str(unread_threads[0])
"<GmailThread len=2 snippet= Do you want to
watch RoboCop this weekend?'>"
>>> len(unread_threads[0].messages)
2
```

```
>>> str(unread_threads[0].messages[0])
"<GmailMessage from='Al Sweigart
<al@inventwithpython.com>' to='Jon Doe
<example@gmail.com>'
timestamp=datetime.datetime(2026, 12, 9, 13,
28, 48)
subject='RoboCop' snippet='Do you want to
watch RoboCop this weekend?'>"
>>> unread_threads[0].messages[0].subject
'RoboCop'
>>> unread_threads[0].messages[0].body
'Do you want to watch RoboCop this weekend?
\r\n'
>>> unread_threads[0].messages[0].timestamp
datetime.datetime(2026, 12, 9, 13, 28, 48)
>>> unread_threads[0].messages[0].sender
'Al Sweigart <al@inventwithpython.com>'
>>> unread_threads[0].messages[0].recipient
'Jon Doe <example@gmail.com>'
```

To retrieve more than the 25 most recent unread emails, pass an integer for the `maxResults` keyword argument. For example, `ezgmail.unread(maxResults=50)` will return the 50 most recent unread emails.

Like the `ezgmail.unread()` function, the `ezgmail.recent()` function will return the 25 most recent threads in your Gmail account:

```
>>> recent_threads = ezgmail.recent()
>>> len(recent_threads)
25
>>> recent_threads =
ezgmail.recent(maxResults=100)
>>> len(recent_threads)
46
```

You can pass an optional `maxResults` keyword argument to change this limit.

Searching for Mail

In addition to using `ezgmail.unread()` and `ezgmail.recent()`, you can search for specific emails, the same way you would if you entered queries into the Gmail search box, by calling `ezgmail.search()`:

```
>>> result_threads =
ezgmail.search('RoboCop')
>>> len(result_threads)
1
>>> ezgmail.summary(result_threads)
Al, Jon - Do you want to watch RoboCop this
weekend? - Dec 09
```

The previous `search()` call should yield the same results as if you had entered *RoboCop* into the search box, as in Figure 20-1.

Figure 20-1: Searching for RoboCop *email at the Gmail website*

Like `unread()` and `recent()`, the `search()` function returns a list of `GmailThread` objects. You can also pass to the `search()` function any of the special search operators that you can enter into the search box, such as the following:

`'label:UNREAD'` For unread email

`'from:al@inventwithpython.com'` For email from *al@inventwithpython.com*

`'subject:hello'` For email with "hello" in the subject

`'has:attachment'` For email with file attachments

You can view a full list of search operators at *https://support.google.com/mail/answer/7190*.

Downloading Attachments

A `GmailMessage` object has an `attachments` attribute that is a list of filenames for the message's attached files. You can pass any of these names to a `GmailMessage` object's `downloadAttachment()` method to download the files. You can also download all of them at once with `downloadAllAttachments()`. By default, EZGmail saves attachments to the current working directory, but you can pass an additional `downloadFolder` keyword argument to `downloadAttachment()` and `downloadAllAttachments()` as well. Here is an example:

```
>>> import ezgmail
>>> threads = ezgmail.search('vacation photos')
>>> threads[0].messages[0].attachments
['tulips.jpg', 'canal.jpg', 'bicycles.jpg']
>>> threads[0].messages[0].downloadAttachment('tulips.jpg')
>>> threads[0].messages[0].downloadAllAttachments(downloadFolder='vacation2026')
['tulips.jpg', 'canal.jpg', 'bicycles.jpg']
```

If a file already exists with the attachment's filename, the downloaded attachment will automatically overwrite it.

EZGmail contains additional features, and you can find the full documentation at *https://github.com/asweigart/ezgmail*.

SMS Email Gateways

People are more likely to be near their smartphones than their computers, so text messages are often a more reliable way of sending immediate notifications than email. Also, text messages are usually shorter, making it more likely that a person will get around to reading them. The easiest, though not most reliable, way to send text messages is by using a *short message service (SMS) email gateway*, an email server that a cell phone provider has set up to receive texts via email and then forward them to the recipient as text messages.

You can write a program to send these emails using EZGmail or the `smtplib` module. The phone number and phone company's email server make up the recipient email address. For example, to send a text

to a Verizon customer with the phone number 212-555-1234, you would send an email to *2125551234@vtext.com*. The subject and body of the email would appear in the body of the text message.

You can find the SMS email gateway for a cell phone provider by doing a web search for "sms email gateway *provider name*." Table 20-1 lists the gateways for several popular providers. Many providers have separate email servers for SMS, which limits messages to 160 characters, and Multimedia Messaging Service (MMS), which has no character limit. If you wanted to send a photo, you would have to use the MMS gateway and attach the file to the email.

If you don't know the recipient's cell phone provider, you can try searching for it using a carrier lookup site. The best way to find these sites is by searching the web for "find cell phone provider for number." Many of these sites will let you look up numbers for free (though they will charge you to look up hundreds or thousands of phone numbers through their API).

Table 20-1: SMS Email Gateways for Cell Phone Providers

Cell phone provider	SMS gateway	MMS gateway
AT&T	*number@txt.att.net*	*number@mms.att.net*
Boost Mobile	*number@sms.myboostmobile.com*	Same as SMS
Cricket	*number@sms.cricketwireless.net*	*number@mms.cricketwireless.net*
Google Fi	*number@msg.fi.google.com*	Same as SMS
Metro PCS	*number@mymetropcs.com*	Same as SMS
Republic Wireless	*number@text.republicwireless.com*	Same as SMS
Sprint (now T-Mobile)	*number@messaging.sprintpcs.com*	*number@pm.sprint.com*
T-Mobile	*number@tmomail.net*	Same as SMS
U.S. Cellular	*number@email.uscc.net*	*number@mms.uscc.net*
Verizon	*number@vtext.com*	*number@vzwpix.com*
Virgin Mobile	*number@vmobl.com*	*number@vmpix.com*
Xfinity Mobile	*number@vtext.com*	*number@mypixmessages.com*

While SMS email gateways are free and simple to use, there are several major disadvantages to them:

- You have no guarantee that the text will arrive promptly, or at all.
- You have no way of knowing if the text failed to arrive.
- The text recipient has no way of replying.
- SMS gateways may block you if you send too many emails, and there's no way to find out how many is "too many."
- The fact that the SMS gateway delivers a text message today doesn't mean it will work tomorrow.

Sending texts via an SMS gateway is ideal when you need to transmit the occasional nonurgent message. If you want a more reliable way to send SMS text messages, especially in bulk, you can use the API of a telecom service provider such as Twilio. These services often require a subscription or usage fees, and you may need to submit an application to use them. Regulations may differ for each country and change over time.

An alternative to sending SMS texts is to use a free push notification service, as the next section will explain.

Push Notifications

HTTP *pub-sub notification services* allow you to send and receive short, disposable messages over the internet via HTTP web requests. Chapter 13 covers the use of the Requests library to make HTTP requests, and we'll use it here to interact with the free online service ntfy (pronounced "notify" and always written in lowercase) at *https://ntfy.sh*. The ntfy service is free and doesn't require any sign-up or registration.

Before we get started, install the ntfy app on your mobile phone so that you can receive notifications. This app is free and can be found in the app stores for Android and iPhone. You can also receive notifications in your web browser by going to *https://ntfy.sh/app*.

These apps check with the ntfy service for any messages sent to a topic. You can think of a topic as a chat room or group chat name. Anyone in the world can send messages to a topic, and anyone in the world can subscribe to a topic to read these messages. If you want to send messages to just yourself, use a secret topic with random letters. Treat this topic name as a password and share it only with those you intend to read the messages. This chapter will use the topic `AlSweigartZPgxBQ42` in the example code, though I recommend you use your own secret topic that contains a suffix of random letters and numbers. Topics are case-sensitive, and even if you keep the topic a secret, do not use ntfy to send sensitive information such as passwords or credit card numbers.

Sending Notifications

Sending a push notification to everyone who is subscribed to a topic requires nothing more than making an HTTP request to the ntfy web server. This means it can be done entirely with the Requests library. You don't need to install a ntfy-specific package.

To send the request, enter the following into the interactive shell. Replace the `AlSweigartZPgxBQ42` example topic used throughout this chapter with your own random, secret topic:

```
>>> import requests
>>> requests.post('https://ntfy.sh/AlSweigartZPgxBQ42', 'Hello, world!')
<Response [200]>
```

Note that we call `requests.post()` to make a POST HTTP request to send a notification. This is different from the `requests.get()` function covered in Chapter 13 to download web pages.

Anyone subscribed to the topic `AlSweigartZPgxBQ42` will receive the message "Hello, world!" within a few seconds (though sometimes messages may be delayed by a few minutes). You can also view these yourself at *https://ntfy.sh/AlSweigartZPgxBQ42*.

The ntfy service has some limitations. Free users are limited to 250 messages per day, and messages can be 4,096 bytes in size at most. Flooding messages to various topics may result in your IP address becoming temporarily blocked. You can obtain a paid account to increase these limits on the ntfy website. Paid ntfy accounts can set reserved topics and limit who posts to them. If you don't have permission to post to a reserved topic, you'll get a `<Response [403]>` response to your `requests.post()` function call.

Within the 4,096-byte limit, your messages can take on any format. Note that there is no way to determine who posted a message to a topic, so you may want to include "To" and "From" labels within the text of your message. Better yet, you could do so using JSON or some other data serialization format covered in Chapter 18.

If you want your Python programs to send you a notification, these are the only two lines of code you need once you have the ntfy app installed on your phone and have subscribed to the topic. You're free to run your Python program and step away for coffee, knowing that you'll receive a notification on your phone when your program has finished its boring task.

Transmitting Metadata

While the message you send is a freeform text string value, ntfy can optionally attach metadata values, such as a title, a priority level, and tags, to each message.

A *title* is similar to an email subject line, and most apps display it above the message text in a larger font. A *priority level* ranges from 1 (the lowest priority) to 5 (the highest), with 3 being the default. A higher priority doesn't deliver messages any faster; it just allows subscribers to configure their notification apps to display only messages of a certain priority or higher. *Tags* are keywords that subscribers can use to filter messages. Tags can also be the name of an emoji to display next to the message title. You can find a list of these emojis at *https://docs.ntfy.sh/publish/#tags-emojis*.

This metadata is included in the HTTP request as headers, so you'll need to pass a dictionary of them to the `headers` keyword argument. Enter the following into the interactive shell to post a message with metadata:

```
>>> import requests
>>> requests.post('https://ntfy.sh/
AlSweigartZPgxBQ42', 'The rent is too
high!',
headers={'Title':'Important: Read this!',
'Tags': 'warning,neutral_face',
'Priority':'5'})
<Response [200]>
```

These features are useful for human users reading the notifications on their phone app. However, you can also write code so that Python scripts can receive push notifications, as we'll discuss in the next section.

Receiving Notifications

Your Python programs can also read the messages posted to a particular topic by making HTTP requests with the Requests library. Send a notification message using the code in the previous sections, and then enter the following into the interactive shell using the same topic as the notifications:

```
>>> import requests
>>> resp = requests.get('https://ntfy.sh/
AlSweigartZPgxBQ42/json?poll=1')
>>> resp.text
'{"id":"1jnHKeFNqwnS","time":1797823340,"expi
res":1797866540,"event":
"message","topic":"AlSweigartZPgxBQ42","messa
ge":"Hello, world!"}\n
{"id":"wZ22cjyKXw1F","time":1797823712,"expir
es":1797866912,"event":
"message","topic":"AlSweigartZPgxBQ42","title
":"Important: Read this!",
"message":"The rent is too
high!","priority":5,"tags":["warning",
"neutral_face"]}\n'
```

Note that we call the `requests.get()` function to receive notifications, unlike the `requests.post()` function used when sending notifications. Also, the URL ends in `/json?poll=1`.

This is retrieving messages through polling, which returns all the messages for a topic that the server has. There are also streaming methods for retrieving ntfy messages, but polling has the simplest code. You can also add a *since* URL parameter after *poll=1* to get the messages by one of the following criteria:

since=10m Retrieves all messages for the topic in the last 10 minutes. You can also use *s* for seconds and *h* for hours, such as *since=2h30m* for all messages in the last two and a half hours.

since=1737866912 Retrieves all messages since the Unix epoch timestamp of `1737866912`. This kind of timestamp is returned by `time.time()` and represents the number of seconds since January 1, 1970. Chapter 19 covers time-related functions.

since=wZ22cjyKXw1F Retrieves all messages after the message that had the ID of `'wZ22cjyKXw1F'`.

Separate additional URL parameters by an ampersand (&). For example, passing the URL *https://ntfy.sh/AlSweigartZPgxBQ42/json?poll=1&since=10m* retrieves all messages for the *AlSweigartZPgxBQ42* topic in the last 10 minutes. To reduce the load on the ntfy server, you should poll only once a minute or once every few minutes rather than as

fast as possible in an infinite loop. If you need to receive notifications immediately, consult the online documentation to read about subscribing to notification streams.

The text of this HTTP response is not valid JSON, since it contains a JSON object on each line of text, rather than one JSON object, so we use the `splitlines()` string method before parsing them individually with the `json` module (as covered in Chapter 18). Continue the previous interactive shell example:

```
>>> import json
>>> notifications = []
>>> for json_text in resp.text.splitlines():
...     notifications.append(json.loads(json_text))
...
>>> notifications[0]['message']
'Hello, world!'
>>> notifications[1]['message']
'The rent is too high!'
```

The `json.loads()` function converts the JSON text from ntfy into a Python dictionary. Let's look at each of the key value pairs:

`"id":"wZ22cjyKXw1F"` The `'id'` key's value is a unique identification string that can help differentiate multiple notifications even if they have the same text.

`"time":1797823712` The `'time'` key's value is a Unix epoch timestamp of when the notification was created. Calling `str(datetime.datetime .fromtimestamp(1797823712))` returns the human-readable string `'2026 -12-20 21:28:32'`.

`"expires":1797866912` The `'expires'` key's value is a Unix epoch timestamp of when the notification will be deleted from the ntfy server.

`"event":"message"` The `'event'` key's value can be either `'message'`, `'open'`, `'keepalive'`, or `'poll_request'`. These event types are explained in the online documentation, but for now, you're probably only interested in `'message'` events.

`"topic":"AlSweigartZPgxBQ42"` The topic part of the URL is repeated in the `'topic'` key's value.

`"title":"Important: Read this!"` If the notification has a title, there will be a `'title'` key with it as a string value.

`"message":"The rent is too high!"` The `'message'` key's value is a string of the notification's text.

`"priority":5` If the notification has a priority, there will be a `'priority'` key with an integer value from 1 to 5.

`"tags":["warning","neutral_face"]` If the notification has tags, there will be a `'tags'` key with it as a list of string values. These string values may be the names of emoji characters to display.

By reading the values in this dictionary, your Python programs can use the Requests library to receive notifications made by users or other Python scripts. The ntfy service is one of the easiest ways to make programs that can communicate with each other over the internet (though keep in mind the limit of 250 messages per day for free users).

Summary

We communicate with each other over the internet and cell phone networks in dozens of different ways, but email and texting predominate. Your programs can communicate through these channels, which gives them powerful new notification features.

As a security and spam precaution, some popular email services like Gmail don't allow you to use the standard SMTP and IMAP protocols to access their services. The EZGmail package acts as a convenient wrapper for the Gmail API, letting your Python scripts access your Gmail account. I highly recommend that you set up a separate Gmail account for your scripts to use so that potential bugs in your program don't cause problems for your personal Gmail account.

Texting is a bit different from email, since, unlike with email, you'll need more than just an internet connection to send SMS texts. You can use SMS email gateways to send texts from an email account, though this requires you to know the phone user's telecom carrier and is not a reliable way to send messages. If you're only sending short messages to yourself, you can use the push notification system at *https://ntfy.sh*, then install the ntfy app on your phone to have your Python scripts send messages to topic subscribers.

With these modules in your skill set, you'll be able to program the specific conditions under which your programs should send notifications or reminders. Now your programs will have a reach that's far beyond the computer they're running on!

Practice Questions

1. When using the Gmail API, what are the credentials and token files?
2. In the Gmail API, what's the difference between "thread" and "message" objects?
3. Using `ezgmail.search()`, how can you find emails that have file attachments?
4. What are some of the disadvantages of using an SMS email gateway to send text messages?
5. What Python library can send and receive notifications to ntfy?

Practice Programs

For practice, write programs to do the following tasks.

Umbrella Reminder

Chapter 13 showed you how to use the `requests` module to scrape data from *https://weather.gov*. Write a program that runs just before you wake up in the morning and checks whether rain is in the forecast for that day. If so, have the program text you a reminder to pack an umbrella before leaving the house.

Auto Unsubscriber

Write a program that scans your email account, finds all the unsubscribe links in all your emails, and automatically opens them in a browser. This program will have to log in to your Gmail account. You can use Beautiful Soup (covered in Chapter 13) to check for any instance where the word *unsubscribe* occurs within an HTML link tag. Once you have a list of these URLs, you can use `webbrowser.open()` to automatically open all of these links in a browser.

You'll still have to manually go through and complete any additional steps to unsubscribe yourself from these lists. In most cases, this involves clicking a link to confirm. But this script saves you from having to go through all of your emails looking for unsubscribe links.

Email-Based Computer Control

Write a program that checks an email or ntfy account every 15 minutes for any instructions you send it and executes those instructions automatically. For example, BitTorrent is a peer-to-peer downloading system. Using free BitTorrent software such as qBittorrent, you can download large media files on your home computer. If you send the program a (completely legal, not at all piratical) BitTorrent link, the program will eventually check its email or look for ntfy notifications,

find this message, extract the link, and then launch qBittorrent to start downloading the file. This way, you can have your home computer begin downloads while you're away, and finish the (completely legal, not at all piratical) download by the time you return home.

Chapter 19 covered how to launch programs on your computer using the `subprocess.Popen()` function. For example, the following call would launch the qBittorrent program, along with a torrent file:

```
qbProcess = subprocess.Popen(['C:\\Program
Files (x86)\\qBittorrent\\
qbittorrent.exe',
'shakespeare_complete_works.torrent'])
```

Of course, you'll want the program to make sure the emails come from you. In particular, you might want to require that the emails contain a password, since it is fairly trivial for hackers to fake a "from" address in emails. The program should delete the emails it finds so that it doesn't repeat instructions every time it checks the email account. As an extra feature, have the program email or text you a confirmation every time it executes a command. Since you won't be sitting in front of the computer that is running the program, it's a good idea to use the logging functions (see Chapter 5) to write a text file log that you can check if errors come up.

The qBittorrent program (as well as other BitTorrent applications) has a feature that enables it to quit automatically after the download completes. Chapter 19 explained how you can determine when a launched application has quit with the `wait()` method for `Popen` objects. The `wait()` method call will block until qBittorrent has stopped, and then your program can email or text you a notification that the download has completed.

There are plenty of possible features you could add to this project. If you get stuck, you can download an example implementation of this program from *https://nostarch.com/automate-boring-stuff-python-3rd-edition*.