Extracted from:

Build Chatbot Interactions

Responsive, Intuitive Interfaces with Ruby

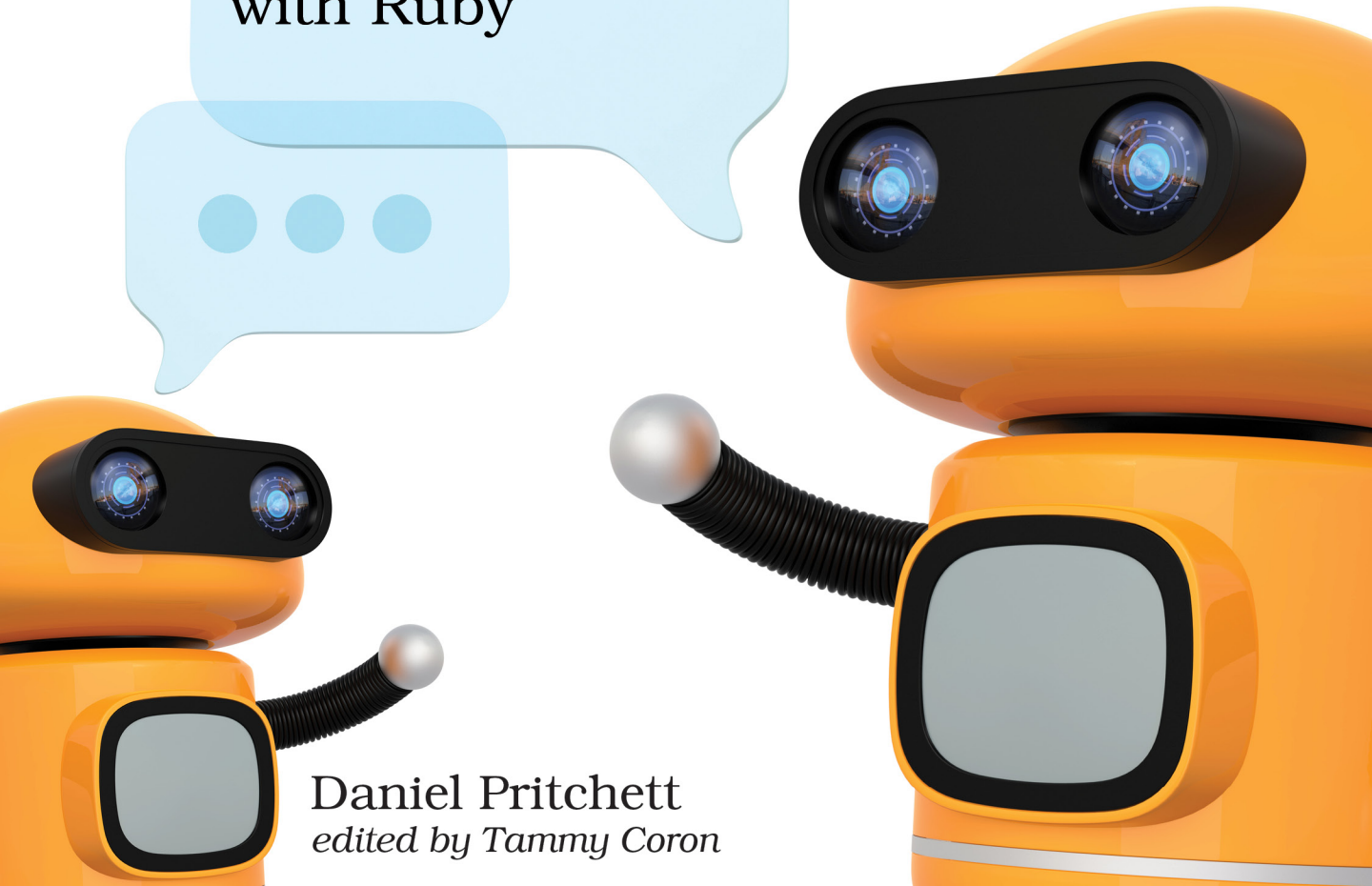
The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Build Chatbot Interactions

Responsive, Intuitive Interfaces with Ruby



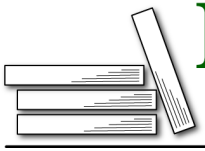
Daniel Pritchett
edited by Tammy Coron

Build Chatbot Interactions

Responsive, Intuitive Interfaces with Ruby

Daniel Pritchett

The Pragmatic Bookshelf
Raleigh, North Carolina



Our Pragmatic books, screencasts, and audio books can help you and your team create better software and have more fun. Visit us at *https://pragprog.com*.

The team that produced this book includes:

Publisher: Andy Hunt
VP of Operations: Janet Furlow
Managing Editor: Susan Conant
Development Editor: Tammy Coron
Copy Editor: Paula Robertson
Indexing: Potomac Indexing, LLC
Layout: Gilson Graphics

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

ISBN-13: 978-1-68050-632-7
Book version: P1.0—June 2019

CHAPTER 11

Send Messages with SMS and Email

As a bot owner or career software developer, you can expect to send and receive plenty of emails, text messages, and other alerts. Because chat is a conversational medium, sending and receiving messages with Lita is a natural fit.

In this chapter, you'll build two similar message-sending Lita skills as you learn how to push messages beyond the confines of your bot and out into the real world. By the end of this chapter, you'll have a firm grasp on two of the most common communication media for automation: emails and SMS messages.

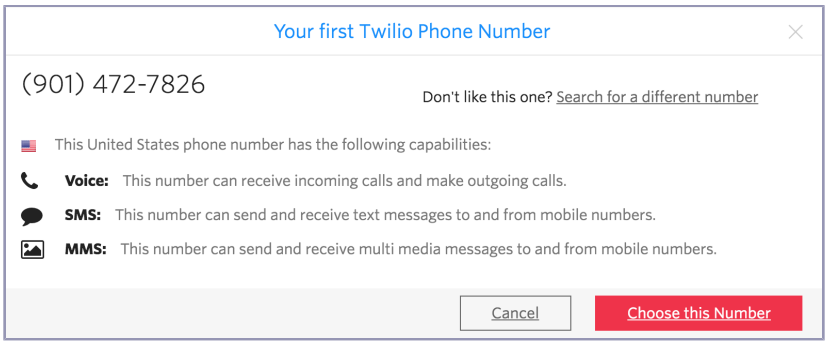
Use Twilio to Send Text Messages

Twilio is a service that provides powerful API access to send and receive phone calls and text messages. The possibilities for group calls and texts, scripted answering services, and voicemail recording are quite broad. Even non-developers are likely to encounter Twilio-based products, though they might not realize it.

Any sort of phone automation you can imagine is plausible with Twilio—from robocalling to answering to recording to re-routing to messaging; it's all in the API. In this section, you get your feet wet by sending text messages through the service. The first step is signing up for a new account at twilio.com.

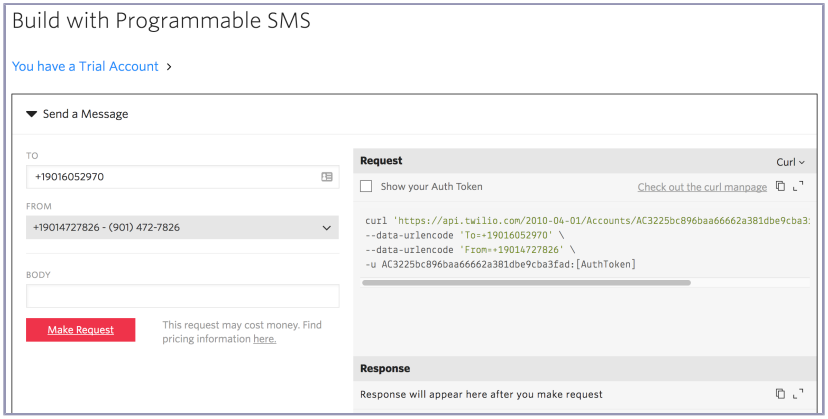
Twilio signup is straightforward—you need an email address and a credit card. Don't worry about a charge yet; Twilio includes promotional credits for first-time users. The promotional credits cover the small monthly fee for renting the phone number that sends your messages, as well as a per-message fee for sending text messages.

Once you're registered, you need to claim a local phone number that Twilio can use for sending outbound messages. The first screenshot on page 6 shows the confirmation dialog for selecting a Memphis-area number.



Send a text message with the Twilio API

Armed with your first Twilio outbound phone number, you can move right into sending your first Twilio-powered text message. Like most of the APIs used in this book, the core is a single HTTP transaction. Twilio's docs are particularly good at guiding you to this first successful transaction, including a `cURL` command tailored to send a message with your newly registered account. Copy and paste the command from your browser—it'll look like the following screenshot—and run it in your terminal.



When you execute the command, you see a JSON response detailing the successful delivery. You should also receive the sample text message at the destination number you supplied.

Here's a sample Twilio response from a successful SMS delivery.

```
# n.b. api keys and message IDs truncated to fit the page
{
  "sid": "SM4625ac9f0cda43c7a36c92e9212c06ef",
  "date_created": "Sun, 19 Nov 2017 20:42:59 +0000",
```

```
  "date_updated": "Sun, 19 Nov 2017 20:42:59 +0000", "date_sent": null,
  "account_sid": "user_id_redacted", "to": "+12135551234",
  "from": "+19014727826", "messaging_service_sid": null,
  "body": "Sent from your Twilio trial account - Hello reader!",
    "status": "queued",
  "num_segments": "1", "num_media": "0", "direction": "outbound-api",
  "api_version": "2010-04-01", "price": null, "price_unit": "USD",
    "error_code": null,
  "error_message": null,
  "uri": "/2010-04-01/Accounts/user_id_redacted/Messages/SMxxx.json",
  "subresource_uris":
    {
      "media": "/2010-04-01/Accounts/id_redacted/Messages/SMxxx/Media.json"
    }
}
```

Your next task is to create a new Lita handler skill that leverages Twilio's Ruby gem to send text messages on demand.

Configure a Twilio handler for Lita

As with every user-facing Lita skill, you need to generate a new handler that listens for a specific pattern of input and responds with some relevant Ruby automation. In this case, your input looks like "send a text message," and the automation shuffles the text message off to Twilio's API.

Start by creating a new Lita handler skill:

```
$ lita handler twilio-texter
```

Switch into the newly created lita-twilio-texter folder and complete the empty parts of your gemspec file. Be sure to add the dependency on twilio-ruby to your own gemspec as demonstrated on line 11.

```
Line 1  # lita-twilio-texter.gemspec
-
-   Gem::Specification.new do |spec|
-     spec.name          = "lita-twilio-texter"
5
-     # ... snipped boilerplate ...
-
-     spec.add_runtime_dependency "lita", ">= 4.7"
-
10    # don't forget this part!
-     spec.add_runtime_dependency 'twilio-ruby', '~> 5.5'
-
-     # ... snipped more boilerplate ...
-   end
```

Build and test the Twilio handler

The Twilio gem's documentation[1] walks you through the basics of setting up a Twilio connection and sending a message from Ruby. Connecting to Twilio requires an account SID and an auth token. You can think of these as a username and password. These credentials are available from the Twilio web UI, and you'll see them as HTTP parameters in the cURL command you just sent to test your first message.

Connect to Twilio from Ruby

Set up the Twilio connection inside the lita-twilio-texter handler file:

```
# lib/lita/handlers/twilio_texter.rb

config :twilio_sid,   default: ENV['TWILIO_ACCOUNT_SID']
config :twilio_token, default: ENV['TWILIO_AUTH_TOKEN']

def client
  @_client ||= Twilio::REST::Client.new(
    config.twilio_sid,
    config.twilio_token
  )
end
```

With this code, you:

- Provide a way for the calling code (your Lita bot) to inject Twilio credentials with either Lita config variables or regular environment variables.
- Create a RESTful Twilio connection object with the :client method.
- Store a connection as a reusable instance variable.
- Return the connection for in-line use.

The code that calls the :client method eventually reads something like this:

```
client.send_text_message(from: me, to: you, body: user_supplied_message).
```

Next, you need a simple test to verify that the returned client object looks correct.

```
# spec/lita/handlers/twilio_texter_spec.rb

describe ':client' do
  it 'should return a logged-in Twilio client' do
    clt = subject.client
    expect(clt.account_sid.empty?).to be_falsey
    expect(clt.auth_token.empty?).to be_falsey
  end
end
```

1. https://github.com/twilio/twilio-ruby

This test verifies that the returned object has a non-blank login and password and implicitly affirms that calls to the :client don't throw an exception. If everything passes, you can be confident that your Twilio client is configured correctly and available for the SMS sending code you're about to write.

Use a Ruby Connection to Send a Twilio SMS

With a properly configured Twilio client object, you're ready to send your first Twilio SMS from within Ruby. Copy the following example code to provide a :send_twilio_sms call that accepts a destination phone number and a message body.

```
# lib/lita/handlers/twilio_texter.rb

def send_twilio_sms(to:, body:)
  response = client.api.account.messages.create(
    from: send_from_number,
    to: to,
    body: body
  )
end

def send_from_number
  client.incoming_phone_numbers.page.first.phone_number
end
```

Note the inclusion of the :send_from_number helper method. This code is necessary because Twilio's API supports high-volume users with a pool of phone numbers available to them. For your bot, however, you only need one, so the code lazily picks the first available number. In a more sophisticated application, you might choose numbers based on proximity to the end user or other business rules.

You can use the following tests to verify that things are working as intended. Note, however, that these tests—as written—are dependent on hard-coded phone numbers. In production, and for more robust tests, you may want to inject known good numbers at runtime or use an HTTP testing library like WebMock.[2]

```
# spec/lita/handlers/twilio_texter_spec.rb

# your phone number goes here
#   twilio insists you text 'verified' numbers during trial
describe ':send_twilio_sms' do
  it 'should work' do
    response = subject.send_twilio_sms(
      to: '+12135551234',
      body: 'hi from lita test')
```

2. https://github.com/bblimke/webmock

```
    expect(response.error_code).to eq(0)
    expect(response.status).to eq("queued")
  end
end

# also dependent on your trial number's area code
describe ':send_from_number' do
  it 'should return my trial number' do
    number = subject.send_from_number
    expect(number.start_with?('+1213')).to be_truthy
  end
end
```

By verifying that the returned objects are at least behaving like properly instrumented Twilio API objects, you can be somewhat sure your bot's ready to go.

Enable Lita users to send SMS messages using a Twilio wrapper

Your Ruby handler code now has an entry point for your Lita skill, and your users' Lita invocation will look like this:

```
Lita text 12135551234 hi mom
```

The following code is similar to other Lita handlers from earlier chapters: this time, it includes a route matcher and a handler method—:send_text—to receive the matches.

```
Line 1  # lib/lita/handlers/twilio_texter.rb
-
-   route /^text\s+(\d+)\s+(.+)$/i,
-     :send_text,
5     command: true,
-     help: { 'text 12135551234 hi mom' =>
-       'texts "hi mom" to a fake number in California' }
-
-   def send_text(message)
10    _, to, body = message.match_data.to_a
-     results = send_twilio_sms(to: to, body: body)
-
-     message.reply "Sent message to #{to}"
-   end
```

Note that on line 3, the matcher expression triggers on the leading word "text," captures the number immediately following, and then captures the remainder of the input as the body of the text message; and on line 10, is a bit of Ruby destructuring (teasing the contents of a data structure apart into multiple variables) to parse the needed parts out of the regex match for safekeeping. The underscore (_) means you're purposefully discarding the first element in the match array.

To verify that things are working, you need some tests.

```
# spec/lita/handlers/twilio_texter_spec.rb

describe ':text' do
  it { is_expected.to route("Lita text 12134441234 hi mom") }
  it { is_expected.to_not route("Lita text hi mom") }

  it 'sends texts' do
    send_message 'Lita text 12135551234 hey daniel'
    expect(replies.last).to eq('Sent message to 12135551234')
  end
end
```

This test asserts that the Lita text [number] [message] handler catches properly formed inputs. It also asserts that Lita's reply to a valid request includes the expected response.