www.phparch.com

July 2016
VOLUME 15 - Issue 7

php[architect]

# harnessing magic


## ALSO INSIDE

# Running Mailing Lists with MailChimp and PHP

Matthew Setter

Despite how old email is—the first email was sent in 1971—and despite the popularity of social media, reports repeatedly show that email is as prevalent as ever. Not only is it still prevalent, but it's far and away more profitable than social media ever was, or likely will ever be. Mailing list services are nicely bundled up in slick web interfaces so that users are required to do as little as possible. That's OK for power users—but we're developers! We like to work with APIs, integrating them into our applications, or sometimes just being able to script up repetitive tasks. Today I'm going to show you how to do that with my current favorite mailing list service—MailChimp.

Given email's popularity, it's not surprising that there is a plethora of online services for managing email campaigns, including such luminaries as MailChimp[1], SendGrid[2], and MailGun[3]. Each of these services, as well as the others in the space, have a range of options, including such features as *lists*, *list segmentation*, *campaigns*, and *automated emails*. Some can even send emails at the same time across multiple time zones—and this is just scratching the surface of what's on offer.

## What is MailChimp?

I've been using MailChimp for several years. I started about four years ago, after hunting around for a service that provided the features I needed, yet at a price point that I could afford. MailChimp did just that.

MailChimp offers all those features, along with customizable email templates, a range of powerful reports, mobile integration, merge tags, integration with WordPress, Facebook, and Eventbrite, and more, all at a very competitive price. Unlike some of its competitors, for the first 2,000 subscribers and up to 12,000 emails per month, you don't pay a cent.

After that, you can elect to start paying for other features, starting at $10 per/month. Not a bad deal. I've enjoyed using the interface to create campaigns for *Master Zend Framework*, the site I started to share my knowledge of Zend Framework.

The UI makes it pretty simple to create lists, send email campaigns, and run reports, but I've been wondering about automating the process for a while. Instead of logging in and filling out the various fields, testing, and scheduling campaigns, what about being able to script it from the command line?

MailChimp has an API that offers a broad range of functionality. This month I want to step you through some of the core functionality and show you how to automate your email campaigns.

Here's what we'll cover:

1. Create an account
2. Create an email list
3. Add several users to the list
4. Update a user's details
5. Create and send an email campaign to our list

To make all this easier, I'm going to use a package I came across recently: mailchimp-api[4], by Drew McLellan. Mailchimp-api is self-described as:

> *A super-simple, minimum abstraction MailChimp API v3 wrapper in PHP.*

I've been experimenting with the library for a little while now and enjoy using it. There's not a lot to remember, with just a few method calls available. This might seem like a shortcoming. But stick with me: you'll see just how flexible the library is as a result.

## The MailChimp API

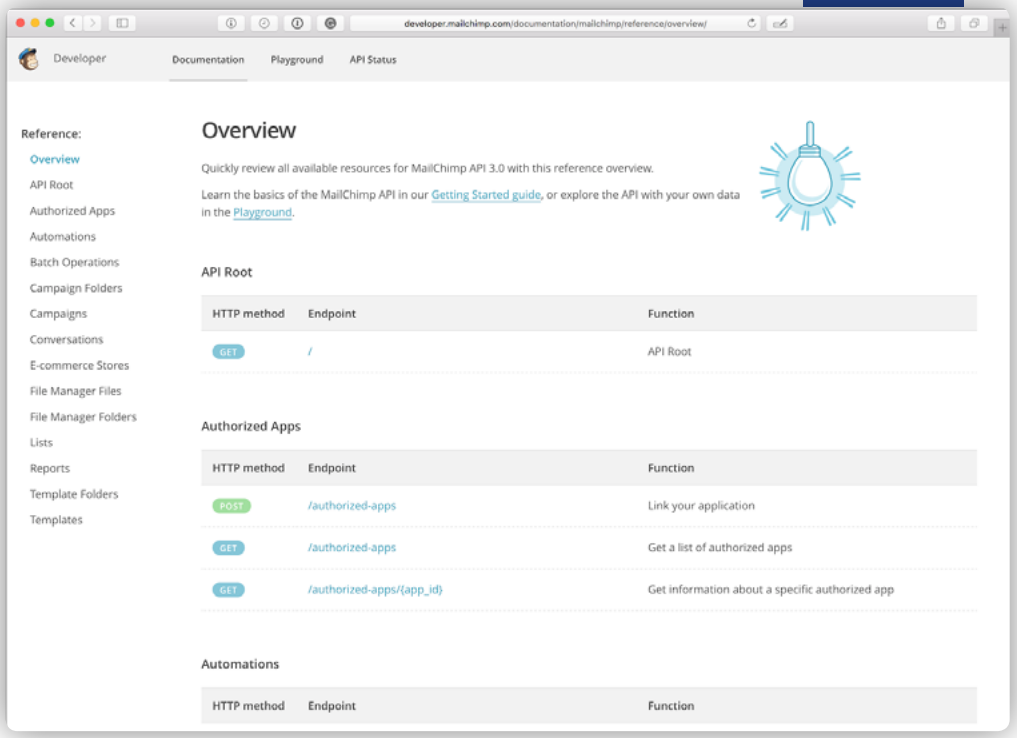
Before we go too much further, let's have a quick look at the API documentation, so that you get a taste of what's on offer. Navigate, in your browser, to the API documentation overview[5]. There you'll see the details for each endpoint, covering:

- The endpoint regular expression
- The request methods that they accept
- A brief description of each one

1 *MailChimp:* http://mailchimp.com
2 *SendGrid:* https://sendgrid.com
3 *MailGun:* https://www.mailgun.com
4 *Mailchimp API:* https://github.com/drewm/mailchimp-api
5 *Mailchimp API documentation overview:* http://phpa.me/mailchimp-api-overview

You can see an example of the Lists endpoint in Figure 1. It summarizes the purpose of the endpoint and the available methods and sub-resources. Then, as you scroll down to each method, you get a detailed description of the arguments that each endpoint accepts, along with properties, if an argument accepts a complex type. This format applies to all the endpoints within the API.

**The MailChimp API documentation** **FIGURE 1**




## Installation

Like all modern PHP libraries, mailchimp-api can be installed using Composer. To do so, from the root of your project directory run the command:

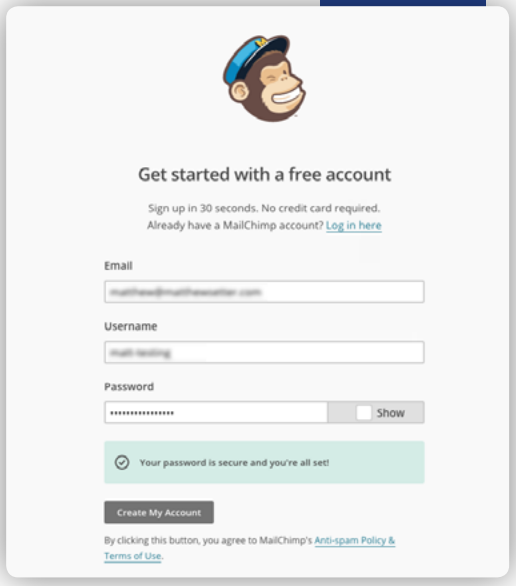
```
composer require drewm/mailchimp-api
```

If you have an existing project, this will add mailchimp-api as a dependency to your project; alternatively, create a `composer.json` file and add it as the first dependency if this is a new project. Regardless of whether this is a new or existing project, the library will be added to the `vendor/` directory.

## Creating a MailChimp Account

With the library installed, you now need a MailChimp account to interact with the API. I'll assume that you don't already have one. If you do, feel free to skip this section. That said, go to the MailChimp signup page[6] in your browser, which you can see in Figure 2, and enter your email address, username, and password.

**MailChimp signup page** **FIGURE 2**



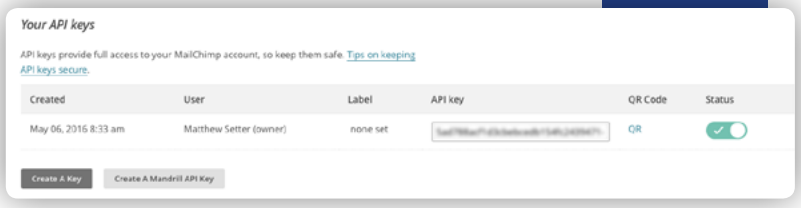

After you've submitted the form, check your email, where you should have received a confirmation email with a link to activate your account. After clicking the link and activating your account, log in[7](https://login.mailchimp.com). You'll see that the account dashboard is pretty bare. You have no campaigns, templates, lists, report data, or automation steps.

## Retrieving Your MailChimp API Key

With your account created, before you can interact with the API you need to create an API key. To do this, click the drop-down in the top right-hand corner, where you see your account image and name, and under that, click **Account**.

On the page that you're directed to, click on **Extra** ˅ and choose **API Keys**. After the page refreshes (you should be at `/account/api/`), you will see a button labeled **Create a Key** under the section **Your API keys**. Click that and you'll be redirected back to the account page, where an API key will have been created, as you can see in Figure 3. You can click in the cell in the **Label** column to give your API key a descriptive name. This also allows you to use different keys with different applications, which comes in handy if you ever need to revoke access to one.

**MailChimp New API Key** **FIGURE 3**




6 *MailChimp signup page: https://login.mailchimp.com/signup*

7 *Mailchimp log in*

Copy the key in the *API key* column into the following code below, replacing `your-api-key`. This code will form the basis for all the code we'll be running in the code examples.

```
<?php

require_once ('vendor/autoload.php');

use \DrewM\MailChimp\MailChimp;

$MailChimp = new MailChimp('your-api-key');
```

## Creating a MailChimp Mailing List

To create a list, we will send a `POST` request to the `/lists` endpoint. You can see in the documentation[8] that there are a number of options available, including: `name`, `contact`, `permission_reminder`, `campaign_defaults`, and `email_type_option`.

**LISTING 1**

```
<?php
require_once ('vendor/autoload.php');
use \DrewM\MailChimp\MailChimp;

$MailChimp = new MailChimp('your-api-key');

$result = $MailChimp->post(
   "lists",
   [
      'name' => 'First List',
      'contact' => [
         'company' => 'My Fun Company',
         'address1' => '123 Anywhere Street',
         'city' => 'Berlin',
         'state' => 'Berlin',
         'zip' => '12203',
         'country' => 'Germany'
      ],
      'permission_reminder' => 'true',
      'campaign_defaults' => [
         'from_name' => 'Matthew Setter',
         'from_email' => 'matthew@example.com',
         'subject' => 'Hey People...!',
         'language' => 'en'
      ],
      'email_type_option' => false,
   ]);

print_r($result);
```

Using just these options, the code in Listing 1 provides:

- A name for the list
- The list's contact details
- Campaign defaults, so that I don't have to provide them every time I create an email campaign
- A reminder about where the users signed up to the list, so that they don't think it's spam
- That the list doesn't support choosing the email format

All being well, you'll see a response similar to the output below, which I've truncated for readability:

```
Array
(
    [id] => 73feddfd00
    [name] => First List
    [contact] => Array
        (
            [company] => My Fun Company
            [address1] => 123 Anywhere Street
            [address2] =>
            [city] => Berlin
            [state] => Berlin
            [zip] => 12203
            [country] => US
            [phone] =>
        )

    [permission_reminder] => true
    [use_archive_bar] => 1
    [campaign_defaults] => Array
        (
            [from_name] => Matthew Setter
            [from_email] => matthew@matthewsetter.com
            [subject] => Hey People...!
            [language] => en
        )

    [notify_on_subscribe] =>
    [notify_on_unsubscribe] =>
    [date_created] => 2016-05-09T14:56:20+00:00

)
```



8 *Mailchimp lists API:* http://phpa.me/mailchimp-lists-api

## Add Users to the Mailing List

Now that we have a mailing list, we need to add some subscribers to the list. Given the modern practice of double opt in [9], this might seem a strange thing to do—that you can add someone directly to your list.

MailChimp gives you the option to do so. Before we jump into the code sample, please make sure you treat this option with the due care it deserves. Otherwise, you might end up with some disgruntled users. With that said, here's how to add a user:

```
$result = $MailChimp->post("lists/bee5472aef/members", [
   'email_address' => 'matthew@matthewsetter.com',
   'email_type' => 'html',
   'status' => 'subscribed',
   'vip' => true
]);

print_r($result);
```

As the list's endpoint documentation shows, we make a post request to the list's endpoint, specifying the list id, and the suffix of `/members`. For the request options, we're going to pass a few of the available options, mainly just the most pertinent: *email_address*, *email_type*, *status*, and *vip*. This will result in a new VIP list member, who will only receive HTML emails.

## Update Users Details

Now, let's imagine that we only had some basic user information, which we provided in the last section, but since then they've provided us with their first and last names. So we're going to add them to the list. To do that, we need to make a patch request and update their details.

As you'd likely expect, the `PATCH` endpoint is only slightly different from the `POST` endpoint, with an MD5 hash of the member's email address being appended to the end (see an example below).

```
$url = sprintf("lists/bee5472aef/members/%s",
               md5('matthew@matthewsetter.com'));
$result = $MailChimp->patch($url, [
   'merge_fields' => ['FNAME'=>'Matthew', 'LNAME'=>'Setter']
]);
```

Here, I've used PHP's `md5` function to create a hash of the email address. If we wanted to, we could also include the hash directly, after retrieving it from the list of members' details, retrieved through a get request as follows

```
$result = $MailChimp->get("lists/bee5472aef/members");
```

Note that in the patch request we only provided the information that we wanted to change. The API is constructed in such a way that `patch` and PUT requests require only the information to be added or changed. All other information will be left intact if not supplied.

## Create and Send an Email Campaign

Now that we have a list with a member, it's time to create a campaign and send an email to it. This is going to take a little bit of work—but not too much. First, we create the campaign, by making a post request to the campaign endpoint (see Listing 2).

Here, I've specified that it will be a normal campaign, and will be sent to the mailing list we created earlier. After that, I specified some

9 *Double opt-in process:* http://phpa.me/mc-double-optin

**LISTING 2**

```
<?php
require_once ('vendor/autoload.php');
use \DrewM\MailChimp\MailChimp;

$MailChimp = new MailChimp('your-api-key');

$result = $MailChimp->post("campaigns", [
    'type' => 'regular',
    'recipients' => [
      'list_id' => 'bee5472aef'
    ],
    'settings' => [
      'subject_line' => 'Here is my subject line',
      'title' => 'here is my title',
      'from_name' => 'Matthew Setter',
      'reply_to' => 'matthew@matthewsetter.com',
    ],
    'tracking' => [
      'opens' => true,
      'html_clicks' => true,
      'google_analytics' => 'regular_campaign_09052016',
    ]
]);
```

settings for the campaign, including the subject line, title, who the email will be from, and a reply to address. The subject line and title are not that compelling. In a real campaign, I'd try to make them something that you'd want to open and read. Finally, I've specified some tracking options.

Whenever I send out a campaign to my lists, I'm always keen to know how many people opened them, and how many took action on the links in the email. Given that, I've set `opens` and `html_clicks` to true. I'm also interested in some Google Analytics data, and have provided a unique code for that as well.

## Create the Email Body

I find this a little strange, but then perhaps it's logical that you have to create the campaign and the campaign email separately. But no matter. To create the body, we have to send a put request to the campaign content endpoint, as in Listing 3.

**LISTING 3**

```
<?php
require_once ('vendor/autoload.php');
use \DrewM\MailChimp\MailChimp;

$MailChimp = new MailChimp('your-api-key');

$html = <<<EOF
<html>
  <head>
    <title>HTML Body</title>
  </head>
  <body>
    Here is the body of the email
  </body>
</html>
EOF;

$result = $MailChimp->put("campaigns/6f23245f05/content", [
  'plain_text' => 'Here is the body of the email',
  'html' => $html
]);
```

Here, we've specified a plain text and HTML body for the email. The content isn't that imaginative, but it's enough for a simple email. To retrieve the id of the campaign, we can make a get request to the campaign's endpoint as follows.

```
$result = $MailChimp->get("campaigns");
```

## Test the Email

Now that the campaign is created and has a body, it's time to send a test email so that we can verify that the email will render as we expect. To do that, we send a post request to the action's test endpoint, as follows.

```
$result = $MailChimp->post(
   "campaigns/6f23245f05/actions/test", [
      'test_emails' => [
         'matthew@matthewsetter.com'
       ],
      'send_type' => 'html'
]);
```

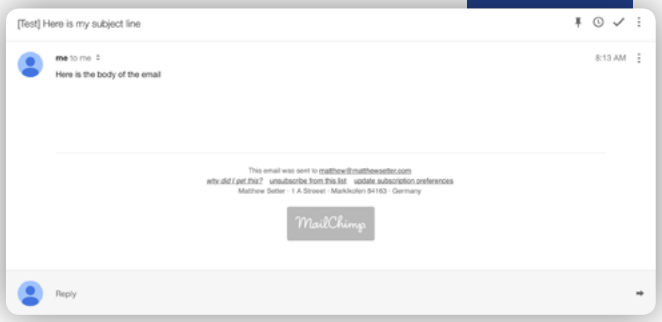
Here, we've specified the campaign id in the endpoint, then for the request data, specified who we're going to send the test email to, and the send type. As we have both a plain text and HTML body, it would make sense to make two requests, testing both formats.

But for a simple example, making one request for the HTML format is enough. After a few minutes, the test email, which you can see in Figure 4, arrived. It's not all that interesting. But it does the job.

If you're interested in creating more imaginative and creative emails, check out the API documentation on email templates[10].

*10 Mailchimp email templates: http://phpa.me/mc-templates*

A test email **FIGURE 4**




## Conclusion

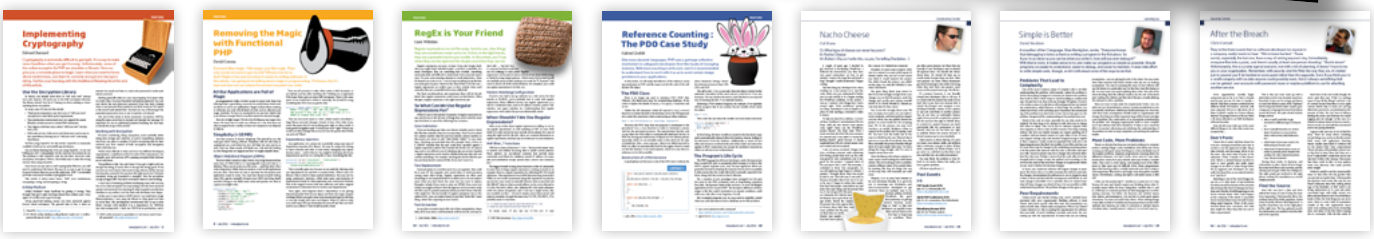
That's how to use Drew McLellan's mailchimp-api library to interact with MailChimp's API. It might seem strange, at first, to use such a simple library to work with such a full-featured API. But, not having to remember a large amount of functionality, you will find working with the API surprisingly simple and remarkably flexible. I've enjoyed using it, as well as exploring the API further. I hope that you take the time to explore it and make the most of it as well.

*Matthew Setter is a software developer specializing in PHP, Zend Framework, and JavaScript. He's also the host of http://FreeTheGeek.fm, the podcast about the business of freelancing as a software developer and technical writer, and editor of Master Zend Framework, dedicated to helping you become a Zend Framework master? Find out more http://www.masterzendframework.com.*

# Want more articles like this one?


Keep your skills current and stay on top of the latest PHP news and best practices by reading each new issue of php[architect], jam-packed with articles.

**Learn more every month about frameworks, security, ecommerce, databases, scalability, migration, API integration, devops, cloud services, business development, content management systems, and the PHP community.**



php[architect]

magazine
books
conferences
training

**www.phparch.com**

**Get the complete issue for only $6!**

We also offer digital and print+digital subscriptions starting at $49/year.