sitepoint

CRAFTING HTML EMAIL

BY RÉMI PARMENTIER

BEAUTIFUL EMAILS THAT WORK EVERYWHERE!

Crafting Effective HTML Email

- **Author:** Rémi Parmentier
- **Series Editor:** Oliver Lindberg
- **Product Manager:** Simon Mackie
- **Technical Editor:** Mark Robbins
- **English Editor:** Ralph Mason
- **Cover Designer:** Alex Walker

Notice of Rights

Notice of Liability

Trademark Notice

sitepoint

Published by SitePoint Pty. Ltd.
10-12 Gwynne St,
Richmond, VIC, 3121
Australia
Web: www.sitepoint.com
Email: books@sitepoint.com

About the Author

Rémi Parmentier is a French front-end developer working at his own small web development agency, Tilt Studio. He loves to learn, and enjoys even more to teach.

This led him on a joyful quest to understand and demystify HTML email coding. Rémi runs workshops, gives talks and writes articles on his blog to help others code better HTML emails.

About SitePoint

Table of Contents

Code Samples

Code in this book is displayed using a fixed-width font, like so:

```
<h1>A Perfect Summer's Day</h1>
<p>It was a lovely day for a walk in the park.
The birds were singing and the kids were all back at school.</p>
```

Where existing code is required for context, rather than repeat all of it, ⋮ will be displayed:

```
function animate() {
   ⋮
new_variable = "Hello";
}
```

Some lines of code should be entered on one line, but we've had to wrap them because of page constraints. An ➥ indicates a line break that exists for formatting purposes only, and should be ignored:

```
URL.open("https://www.sitepoint.com/responsive-web-
➥design-real-user-testing/?responsive1");
```

Tips, Notes, and Warnings

Hey, You!

Tips provide helpful little pointers.

Ahem, Excuse Me ...

Notes are useful asides that are related—but not critical—to the topic at hand. Think of them as extra tidbits of information.

Make Sure You Always ...

... pay attention to these important points.

Watch Out!

Warnings highlight any gotchas that are likely to trip you up along the way.

Chapter

1

Crafting HTML Emails: Coding HTML Emails: A Modern Perspective

Rémi Parmentier

HTML email is a part of pretty much every project, but even many of the most experienced web developers dread having to work with it. This series of tutorials will explore the ins and outs of coding modern HTML emails, showing you how to love the craft rather than fear it.

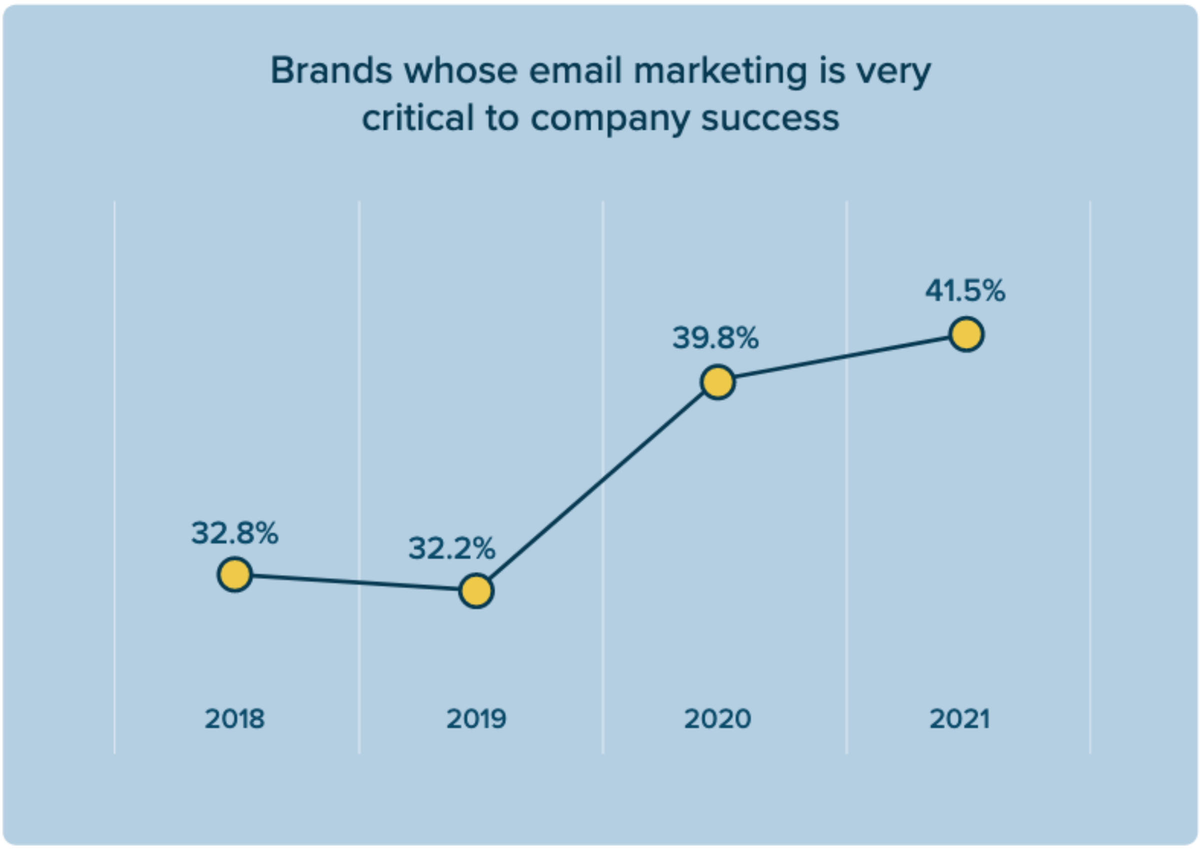
Email Isn't Dead

Email was born in 1971. And it died in 2007, according to FastCompany. Or in 2009, according to The Wall Street Journal. Or maybe it was in 2011, according to Mark Zuckerberg. No one really seems to know.

What we do know, however, is that email is still highly effective:

- "Email drives an ROI of $36 for every dollar spent", according to email marketing platform Litmus.
- Four billion people use email daily, according to data platform Statista.
- In 2021, 41.5% of brands interrogated in the Litmus State of Email report consider email marketing to be critical to company success—an 8.7% increase from just three years earlier.

The following diagram shows the percentage of brands for whom email marketing is critical to success, rising from 32.8% in 2018 to 41.5% in 2021.



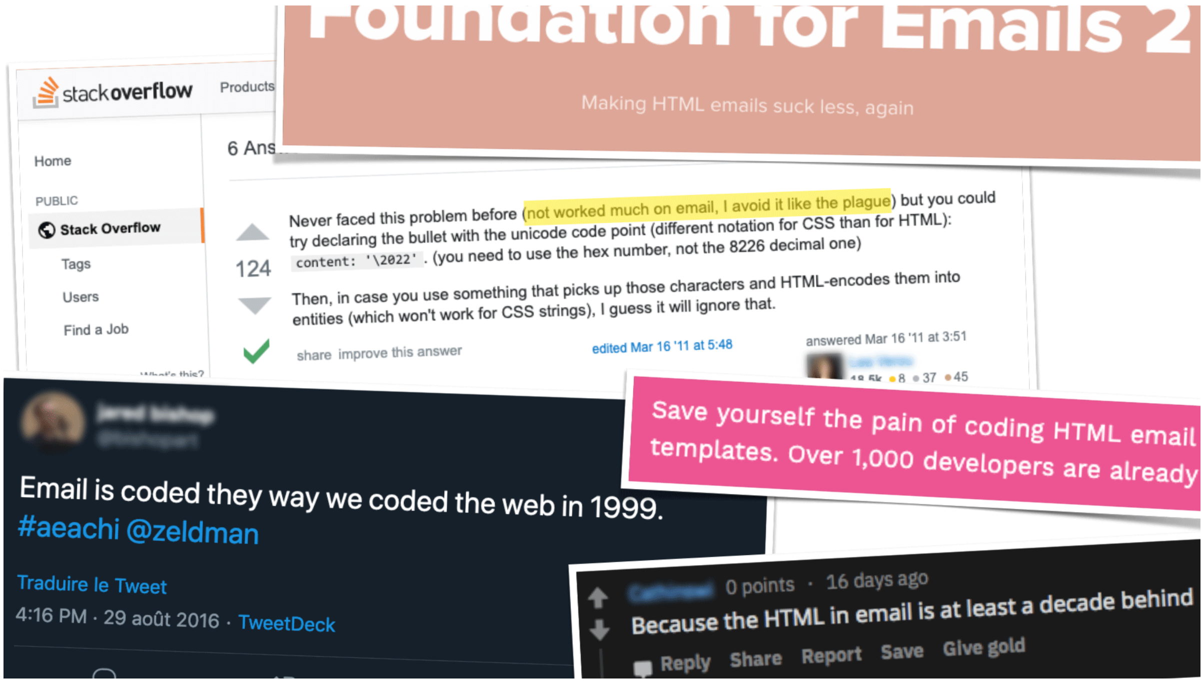
1-1. Brands whose email martketing is critical to company success, rising from 32.8% in 2018 to 41.5% in 2021

According to Harvard law professor Jonathan Zittrain (quoted by The Atlantic), one reason email is still so strong after all these years is that “Email is the last great unowned technology”. This arguably makes it more resilient and robust than any proprietary technology. So it certainly seems that email is here to stay!

HTML Emails Aren’t Stuck in the 1990s

Discussions surrounding the coding of HTML emails on social networks never fail: someone will always snarkily comment on how HTML emails are stuck in the 1990s. And there’s nothing that irritates me more—because it’s completely wrong!

A collection of snarky comments is pictured below, sourced from my 2019 “Think Like an Email Geek” presentation.


stackoverflow Products

Home

PUBLIC

Stack Overflow

Tags

Users

Find a Job

6 Ans

Foundation for Emails 2

Making HTML emails suck less, again

124

Never faced this problem before (not worked much on email, I avoid it like the plague) but you could try declaring the bullet with the unicode code point (different notation for CSS than for HTML): `content: '\2022'`. (you need to use the hex number, not the 8226 decimal one)

Then, in case you use something that picks up those characters and HTML-encodes them into entities (which won't work for CSS strings), I guess it will ignore that.

share improve this answer edited Mar 16 '11 at 5:48 answered Mar 16 '11 at 3:51 [redacted]

[redacted]

Email is coded they way we coded the web in 1999. #aeachi @zeldman

Traduire le Tweet

4:16 PM · 29 août 2016 · TweetDeck

Save yourself the pain of coding HTML email templates. Over 1,000 developers are already

[redacted] 0 points · 16 days ago

Because the HTML in email is at least a decade behind

Reply Share Report Save Give gold

1-2. “Email is coded the way we coded the web in 1999” and other snarky comments about HTML emails

When I code an HTML email, I use plenty of modern styles: media queries (including ones for dark mode), CSS gradients and background images, Flexbox, and even CSS Grid. I also use semantic HTML: heading elements (`<h1>` , `<h2>` , …), paragraphs (`<p>`), and lists elements (`<ul>` , `<li>`).

But I also use tables—because it’s still largely recommended to use tables for layout in HTML emails. But the one and only reason for this boils down to one single word: Outlook. Or, rather, three words: Outlook on Windows. (The other Outlook apps—on macOS, iOS or Android—use either WebKit or Blink and are really fine.) In 2007, Microsoft decided to switch the underlying rendering engine of its mail application from Internet Explorer to … Word. And Word is not really good at rendering HTML and CSS. So the most robust and safest way to tame Word is to use tables.

Still, this doesn’t mean we have to impose tables on everyone, in every email client. In 2015, email developer Nicole Merlin published a foundational article detailing how to create a responsive email without media queries, using `<div>` s and conditional comments for Outlook on Windows.

The world of email clients has kept moving ever since. Later that year, Yahoo fixed a bug preventing the use of media queries. In 2016, Gmail followed along and added official support for media queries in almost all its email clients. Outlook.com has been rebuilt, twice—first in 2015, then again in 2018. Apple Mail has never ceased enriching its rendering engine with things we barely use even on the Web, from `backdrop-filter()` to CSS Scroll Snap.

In 2018, JavaScript partially made its entry into emails via dedicated frameworks (Adaptive Cards for Outlook clients, and AMP for Email in Gmail, AOL, Yahoo Mail and Mail.ru). And then, in 2019, email clients turned to the dark side with the arrival of dark mode and partial support for `@media (prefers-color-scheme)` media queries. The Gmail apps (on iOS and Android) even got an auto dark theme feature, years before Chrome got its own.

Defining an Email Client

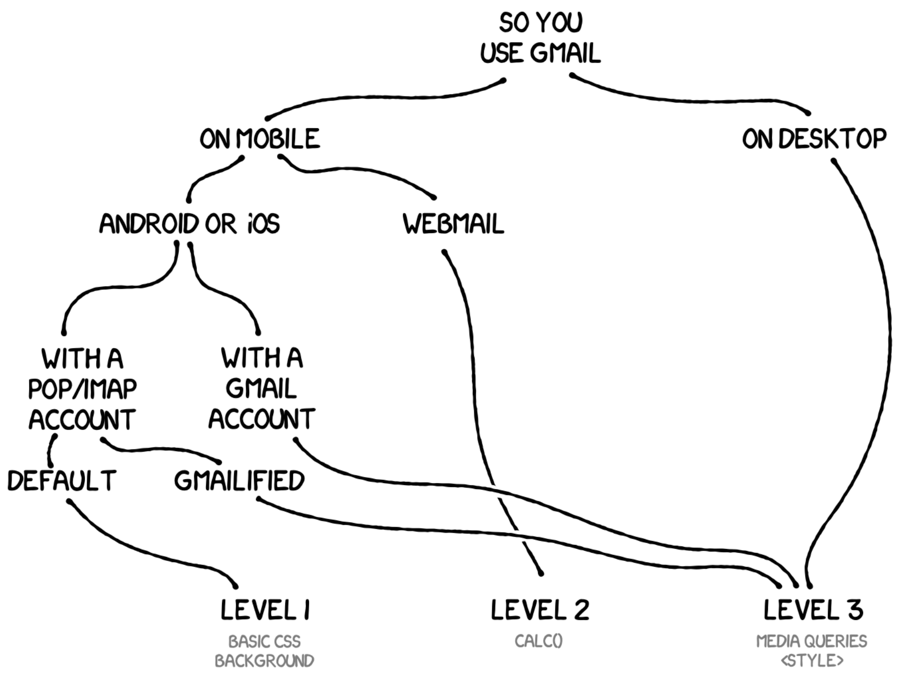
Outlook is not an email client. Neither is Gmail. Nor is Yahoo. Yet I hear angry bosses or helpless developers complain about their email being broken in Outlook all the time. So what's wrong with this?

The thing is, rather than being an email client, Outlook is a brand name. It's a group of different email clients. There's Outlook on Windows, Outlook on macOS, Outlook on iOS, Outlook on Android, and Outlook.com on the Web. Each of these email clients has its own rendering engine (for example, Word for Outlook on Windows, WebKit for Outlook on macOS), its own filters and HTML and CSS restrictions, its own set of features, and its own bugs.

It's a similar situation with Gmail. There's Gmail's desktop webmail, its mobile webmail, its iOS app, and its Android app. Gmail's webmail versions use the rendering engine of your browser (Blink for Chrome, WebKit for Safari, Gecko for Firefox). The mobile apps use the operating system's default web view engine (WebKit for iOS, Blink for Android). But then, each of these email clients has its own variation of HTML and CSS support. And the most egregious example of this is that, in Gmail's mobile apps (on iOS and Android), you get one level of CSS support if you're using a Gmail account (with an `@gmail.com` email address or on your own Google Workspace), but a different level of support if you're using a third-party email address (like an `@outlook.com` or `@yahoo.com` email address). And in the latter case, that's pretty much the worst level of support possible, with no `<style>` tags and no media queries!

A few years ago, I started drawing diagrams of CSS support across all Gmail clients to get a better understanding of this support and better explain it to colleagues and fellow email developers. And while things have certainly improved in the past decade for Gmail, it's still something unexpected and surprising for newcomers.

The diagram below shows the different levels of CSS support for Gmail clients.



1-3. The different levels of CSS support for Gmail clients

In 2017, email marketer Chad White estimated that every email "has approximately 15,000 potential renderings (and that's using conservative math)". And that was before dark mode, and not even taking accessibility into account. Personally, I consider that each recipient of an email will have a unique way of viewing it.

So if you're writing a brief for an agency about which email clients you want to be supported, or if you're giving feedback to your email developer, or if you're developing software related to email clients, make sure to always provide each client's full name, together with its family, its platform, and (when available) its version.

"Email Developer" Is a Job

When I started web development as an amateur in the late 1990s, the Web was all the rage. It was so exciting to be able to publish your own web pages, laid out with bare HTML and maybe just a touch of CSS. Later generations of developers got excited over mobile platforms (in the late 2000s), and then over JavaScript frameworks (in the late 2010s). But no one really ever got excited about being an email developer. No one really wakes up one day thinking "I can't wait to code HTML emails". Even if you've looked keenly at newsletters from a designer's or author's perspective, you probably haven't done so from a developer's one. You don't really plan to become an email developer. It sort of happens to you.

Email marketer Justine Jordan captured this perfectly when she popularized this saying:

> *You don't choose email. Email chooses you.*

It was even turned into a T-shirt by email developer Anne Tomlin.

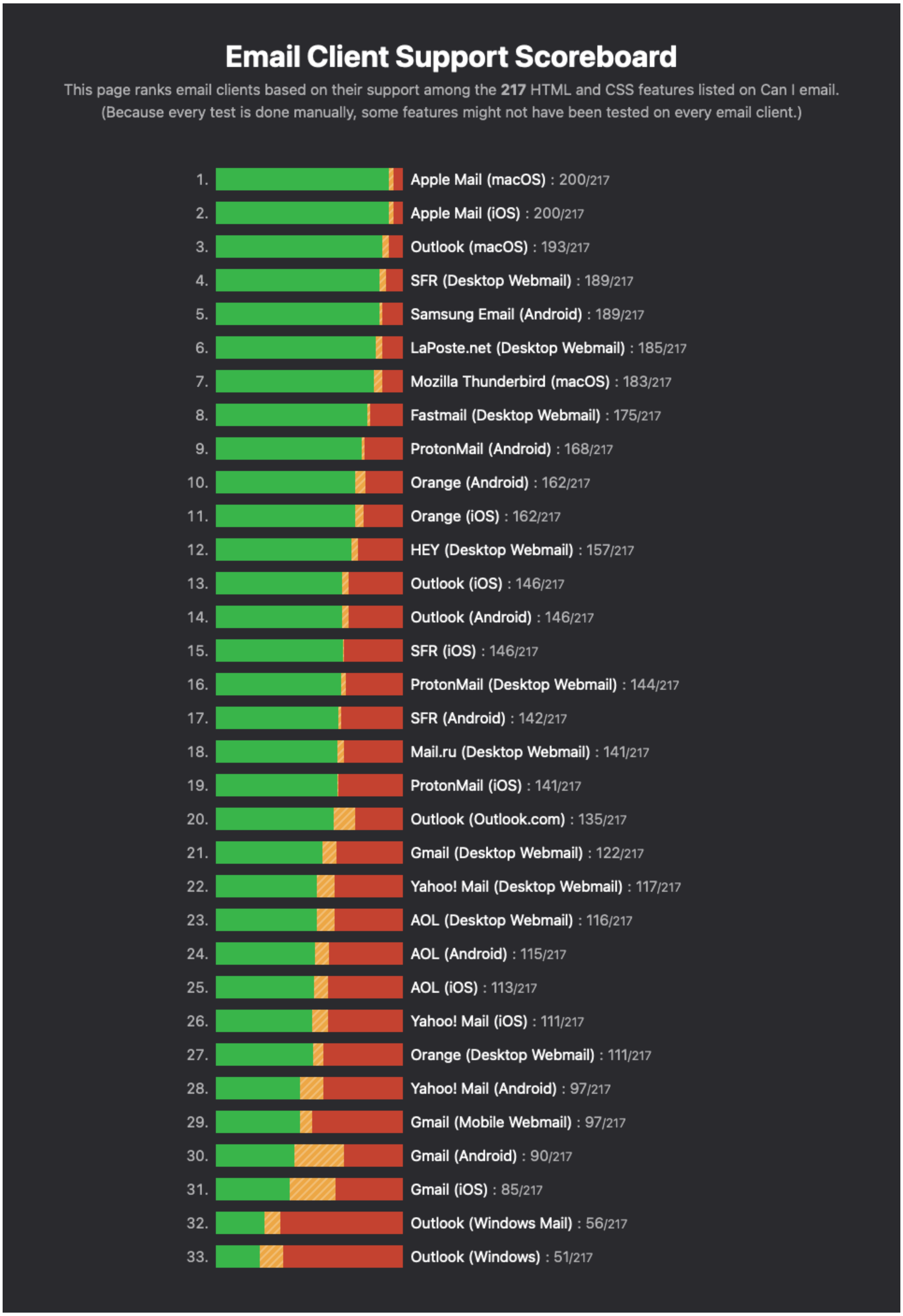
1-4. A T-shirt reading "I didn't choose email. Email chose me."

The role of email developer is a job. And it's one that I'm very proud of.

It's one of the last refuges for those interested in HTML and CSS, but not really JavaScript. And it's a great area to focus on for those who code and care about interoperability, accessibility, maintainability, and graphic design.

Email development is a craft of its own. We should embrace its quirks and weirdness and use its unique abilities. And we should stop comparing it with web development. It's a trap to believe that because you can use HTML and CSS in emails, you should be able to use any HTML and CSS.

Email client support for HTML and CSS is a spectrum that spans from Apple Mail at the top—which supports 200 HTML and CSS features—to Outlook on Windows at the very bottom—which supports only 51, as shown in the Email Client Support Scoreboard from *Can I email*.



Email Client Support Scoreboard

This page ranks email clients based on their support among the **217** HTML and CSS features listed on Can I email.
(Because every test is done manually, some features might not have been tested on every email client.)

1. Apple Mail (macOS) : 200/217
2. Apple Mail (iOS) : 200/217
3. Outlook (macOS) : 193/217
4. SFR (Desktop Webmail) : 189/217
5. Samsung Email (Android) : 189/217
6. LaPoste.net (Desktop Webmail) : 185/217
7. Mozilla Thunderbird (macOS) : 183/217
8. Fastmail (Desktop Webmail) : 175/217
9. ProtonMail (Android) : 168/217
10. Orange (Android) : 162/217
11. Orange (iOS) : 162/217
12. HEY (Desktop Webmail) : 157/217
13. Outlook (iOS) : 146/217
14. Outlook (Android) : 146/217
15. SFR (iOS) : 146/217
16. ProtonMail (Desktop Webmail) : 144/217
17. SFR (Android) : 142/217
18. Mail.ru (Desktop Webmail) : 141/217
19. ProtonMail (iOS) : 141/217
20. Outlook (Outlook.com) : 135/217
21. Gmail (Desktop Webmail) : 122/217
22. Yahoo! Mail (Desktop Webmail) : 117/217
23. AOL (Desktop Webmail) : 116/217
24. AOL (Android) : 115/217
25. AOL (iOS) : 113/217
26. Yahoo! Mail (iOS) : 111/217
27. Orange (Desktop Webmail) : 111/217
28. Yahoo! Mail (Android) : 97/217
29. Gmail (Mobile Webmail) : 97/217
30. Gmail (Android) : 90/217
31. Gmail (iOS) : 85/217
32. Outlook (Windows Mail) : 56/217
33. Outlook (Windows) : 51/217

1-5. Apple Mail on macOS supports 200 HTML and CSS features while Outlook on Windows only supports 51

If you're ready to face the unexpected, I can assure you coding emails will become a joyride—even if a wild one!

Conclusion

Our journey into the world of HTML emails is just getting started. In the following tutorials, we'll cover a lot of tips and tricks for tackling the vagaries of email clients, and also for making our emails more accessible and even interactive!

Although we're going to stay focused on the coding and development side of HTML email, there's a whole lot more to explore around this topic—from design to strategy and deliverability. Here are three of my favorite links for diving deeper into all that:

- Really Good Emails. A collection of over 9,000 curated emails from the world's top companies. And most emails also come with their HTML code!
- Email Resources by email developer Avi Goldman. Features over 400 up-to-date links to resources about email, from copywriting and design to code and deliverability.
- The #emailgeeks Slack channels. Over 12,000 email professionals chat, share, and help each other all week long across dozens of channels (covering code, design and marketing). You may find me over there quite often in the `#email-code` channel!

In our next tutorial in this series, we'll review some of the most important tips and tricks for coding robust emails across email clients.

Chapter

2 Essential Best Practices

Rémi Parmentier

One of the most difficult aspects of HTML email coding is that every email client has its own quirks and features. Email clients usually end up with these quirks through the best of intentions. For example, they might turn a plain text website address into a clickable link. There's also the issue of security. Email clients need to make sure that our email's HTML and CSS won't interfere with their own interface's HTML and CSS. A malicious email could use certain CSS properties (like absolute positioning) to lure people into clicking on overlaid hidden links. So email clients *should* parse, filter and manipulate HTML email code. But this means that we, as email developers, must be aware of this and make our code as friendly as possible for them.

From features to bugs, here are some of the most popular tips and tricks we need to know.

Supporting the Outlooks

According to email analytics tool Litmus, Outlook (on both Windows and macOS) accounted for 4.44% of email client market share in January 2022. That may not seem much, and remember to take email analytics with a pinch of salt, but chances are you'll meet Outlook on Windows at some point in your email developer journey.

Here's what you need to know to make your HTML emails work painlessly in the Outlooks on Windows.

How the Outlook Rendering Engine Works

Since 2007, the Outlooks on Windows have used Word as the rendering engine for HTML and CSS. Microsoft justified the use of Word in 2009:

> *We've made the decision to continue to use Word for creating e-mail messages because we believe it's the best e-mail authoring experience around, with rich tools that our Word customers have enjoyed for over 25 years.*

Not only is Word not very good for rendering HTML and CSS, but the documentation on the matter is abysmal. The only official existing documentation about Word's rendering is a 2006 post from Microsoft explaining HTML and CSS rendering capabilities in Outlook 2007.

It includes information on the following:

- **CORE**: `color` , `background-color` , text properties (`font` , `font-family` , `font-style` , `font-`

`size`, `font-weight`, `text-decoration`, `text-align`, `vertical-align`, `letter-spacing`, `line-height`, `white-space`), border shorthand properties (`border`, `border-color`, `border-style`, `border-width`, `border-collapse`) and a few others.
- **COREEXTENDED**: `text-indent` and margin properties (`margin`, `margin-left`, `margin-right`, `margin-top`, `margin-bottom`).
- **FULL**: `width`, `height`, `padding` (as well as `padding-left`, `padding-right`, `padding-top`, `padding-bottom`) and border longhand properties (`border-left`, `border-left-color`, `border-left-width`, `border-left-style`, and so on).

And each of these categories will only apply to certain HTML elements:

- `<body>` and `<span>` only support **CORE** properties.
- `<div>` and `<p>` support both **CORE** and **COREEXTENDED** properties.
- All the other elements supported by Outlook (like `<table>`, `<td>`, `<h1>`, `<ul>`, `<li>`, and so on) support **CORE**, **COREEXTENDED** and **FULL** properties.

This means we must think about which element to use to apply certain styles. So if we have to define a `width` or a `height` on a generic container element, we'll use a `<table>`. If we need `padding`, we'll also use a `<table>` and a `<td>`.

To this day, the Outlooks on Windows are the sole reason we still use tables in HTML emails. Luckily, there are ways for us to only make those tables visible for Outlook, hiding them from more capable email clients and allowing us to use more semantic code.

Conditional Comments

Microsoft introduced conditional comments back in 1999 in Internet Explorer 5. They were quite popular on the Web during the IE6–9 era, but they were removed for Internet Explorer 10 and 11. Here's how it works: inside a regular HTML comment (`<!-- -->`), you can code a condition that will make the rest of the content visible if that condition is fulfilled. Here's an example:

```
<!--[if IE]>
<p>This is only visible in Internet Explorer.</p>
<![endif]-->
```

It turns out that conditional comments are also supported in the versions of Outlook on Windows using Word's rendering engine. Instead of using `IE` as a condition, we're going to use the `mso` keyword:

```
<!--[if mso]>
<p>This is only visible in Outlook 2007 and above on Windows.</p>
<![endif]-->
```

Conditions can also be tied to a version number. For example, `mso 12` targets Outlook 2007. Unfortunately, Microsoft has stopped incrementing this version number in the latest Outlook releases. So `mso 16` targets both Outlook 2016 and the most recent Outlook 2019.

We can also use operators like `gte` (*greater than or equal*) or `lte` (*less than or equal*) to create more complex conditions:

```
<!--[if gte mso 12]>
<p>This is visible in Outlook 2007 and above.</p>
<![endif]-->
```

Another useful operator is the NOT operator (`!`), which lets us insert code for every email client *except* Outlook on Windows:

```
<!--[if !mso]><!-->
<p>This is visible <p>every email client except the Outlooks on Windows.</p>
</p><!--<![endif]-->
```

`mso-` Properties

Outlook on Windows has hundreds of proprietary CSS properties, mostly recognizable thanks to the `mso-` prefix. A complete list is available in a `.chm` file), but email developer Stig Morten Myre has a handy archive of it readable online.

For a lot of standard CSS properties, Microsoft has an equivalent proprietary version prefixed by `mso-` and suffixed by `-alt`. For example, we can define a `padding` value for just Outlook on Windows with the `mso-padding-alt` property. One way I often use this is when I code buttons. Ideally, the entire visible area of a button should be clickable, so I normally add `padding` to `<a>` elements. But Outlook on Windows doesn't support this. So instead, I wrap each `<a>` element with a `<table>` and apply a `padding` only for Outlook on Windows with the `mso-padding-alt` property:

```
<table border="0" cellpadding="0" cellspacing="0" role="presentation" align="center"
  style="margin:0 auto; max-width:100%; background:#2ea44f; border-radius:5px;
  border-collapse:separate;" class="email-btn">
    <tr>
```

```
        <td style="mso-padding-alt:14px 16px; text-align:center;">
            <a style="padding:14px 16px; display:block; min-width:128px; color:#fff;
            font:bold 16px/20px Helvetica Neue, Roboto, sans-serif; text-decoration:none;
            text-align:center;" href="https://www.example.com" target="_blank" >Call to Action</a>
        </td>
    </tr>
</table>
```

Outlook also has other unprefixed proprietary properties that can mimic their more modern CSS equivalents. For example, `text-underline-color` in Outlook is the same as `text-decoration-color` in CSS. So if you want to apply a specific color to a text underline, you can use both properties:

```
text-underline-color: red; /* Outlook version */
text-decoration-color: red; /* Standard version for clients that supports it */
```

Stig Morten Myre has a great article explaining how to fix bugs with Outlook specific CSS, including tips on how to use `mso-text-raise` or `mso-line-height-rule: exactly`.

VML

VML is SVG's ancestor, crafted by Microsoft in the late nineties. Just like SVG, you can draw content with markup code. For example, if we want to draw a red rectangle, we can use the `<v:rect>` element and the following code:

```
<v:rect xmlns:v="urn:schemas-microsoft-com:vml"
        fillcolor="red"
        stroked="false"
        style="width:200px; height:100px;">
</v:rect>
```

A prerequisite for getting VML to work in Outlook on Windows is to add its namespace declaration (`xmlns:v="urn:schemas-microsoft-com:vml"`). It can either be repeated inline for each VML element we use, or it can be added on the `<html>` element only once. And because VML will only work in Outlook on Windows, we'll make sure to wrap it in a conditional comment. Here's a full working example for our previous red rectangle:

```
<!DOCTYPE html>
<html lang="en" xmlns:v="urn:schemas-microsoft-com:vml">
<head>
    <title>VML rectangle</title>
</head>
```

```
<body>
    <!--[if mso]>
    <v:rect fillcolor="red" stroked="false" style="width:200px; height:100px;"></v:rect>
    <![endif]-->
</body>
</html>
```

Of course, we can do more exciting things than red rectangles. One way we can use VML is to fake properties unsupported by Outlook on Windows—such as background images. If we want to show a background image on our `<body>` element, we can use the `<v:background>` element. However, this might be a source of accessibility issues, as it turns the content within it into part of the VML image, which might get lost for assistive technologies like screen readers.

Campaign Monitor's Backgrounds.cm and Buttons.cm make extensive use of VML to fake background images or rounded corners. And email developer Mark Robbins provides great examples of how VML can be used to create CSS triangles or fake absolute positioning.

Microsoft's "How to Use VML on Webpages" and "VML Object Model Reference" are great places to start learning about VML.

Rendering at `120dpi`

On certain Windows configurations, Outlook on Windows applies DPI scaling on emails. Email developer Courtney Fantinato has a detailed guide to correct Outlook DPI scaling issues. Here are the three rules you need to follow:

1. Add the Microsoft Office namespace on the `<html>` element:

```
<html xmlns:o="urn:schemas-microsoft-com:office:office">
```

2. Add the following `OfficeDocumentSettings` declaration inside the `<head>` element:

```
 <!--[if mso]>
<xml>
  <o:OfficeDocumentSettings>
    <o:PixelsPerInch>96</o:PixelsPerInch>
  </o:OfficeDocumentSettings>
</xml>
<![endif]-->
```

3. Always use dimensions defined in CSS instead of HTML attributes:

```
 <!-- Bad example -->
<table align="center" role="presentation" width="600">…</table>

<!-- Good example -->
<table align="center" role="presentation" style="width:600px;">…</table>
```

The only exception for this is with images. In Outlook on Windows, a width set in a `style` attribute is ignored on images.

Making Your Emails Work without `<style>`

When it comes to applying styles to any HTML content (be it for a web page or an HTML email), there are three ways to do it, using:

- a `<link>` element and an external stylesheet
- a `<style>` element
- an inline `style` attribute

Email clients are very opinionated when it comes to this, and support for each of these techniques can vary wildly. This is usually due to security issues. Email clients need to be very cautious about the styles they allow, because a malicious email could use styles to deceive a user. For example, using `fixed` or `absolute` positioning could let a malicious email developer stack fake elements over an email client's own interface.

Email clients employ a variety of methods for getting around this, such as only allowing a subset of properties or values from a safe list of their own. They also apply prefixing to CSS selectors in an HTML email to prevent email styles from impacting the client's interface. For example, a selector like `.button {}` would become `.rps_1234 .x_button {}` in Outlook.com.

The `<link>` element is very well supported in desktop native applications like Apple Mail (on macOS or iOS) or Outlook (on Windows or macOS). However, it's almost universally ignored by webmail clients (such as Gmail, Outlook.com, Yahoo Mail, and so on) as well as a lot of mobile apps (like Gmail, Outlook or Yahoo Mail, on either iOS or Android). So it's not a recommended way to style an HTML email. But it can have interesting use cases, like when Litmus created a live dynamic Twitter feed in 2015.

The `<style>` element has way better support. But it also has a few quirks. Gmail clients, in particular, only support style tags defined in the `<head>` (not in the `<body>`). They're also very picky about any syntax error, and Gmail will remove an entire `<style>` that contains something it doesn't like (like an `@` in an `@` declaration, for example). It's a common practice to use multiple `<style>` elements in HTML emails. After removing the unsupported ones, Gmail will combine them into a single one that's limited to 16KB.

But `<style>` elements are not always supported. For example, if you receive an email in the Gmail app (on iOS or Android) with a non-Gmail address (like an Outlook.com or Yahoo address), you won't get `<style>` support.

`<style>` elements can also be removed contextually. For example, when you forward an email in the desktop webmail of Gmail, all `<style>` tags are removed. Gmail also removes `<style>` tags when an email is viewed in its unclipped version.

So as a general rule, it's safer and more robust to use inline styles via the HTML `style` attribute. We can also use `<style>` tags, but only as a progressive enhancement, especially for things like `@media` queries or `:hover` pseudo-classes that can't be inlined.

"Making an email work" without the `<style>` element can mean a lot of different things. But it's best to think first and foremost about the following:

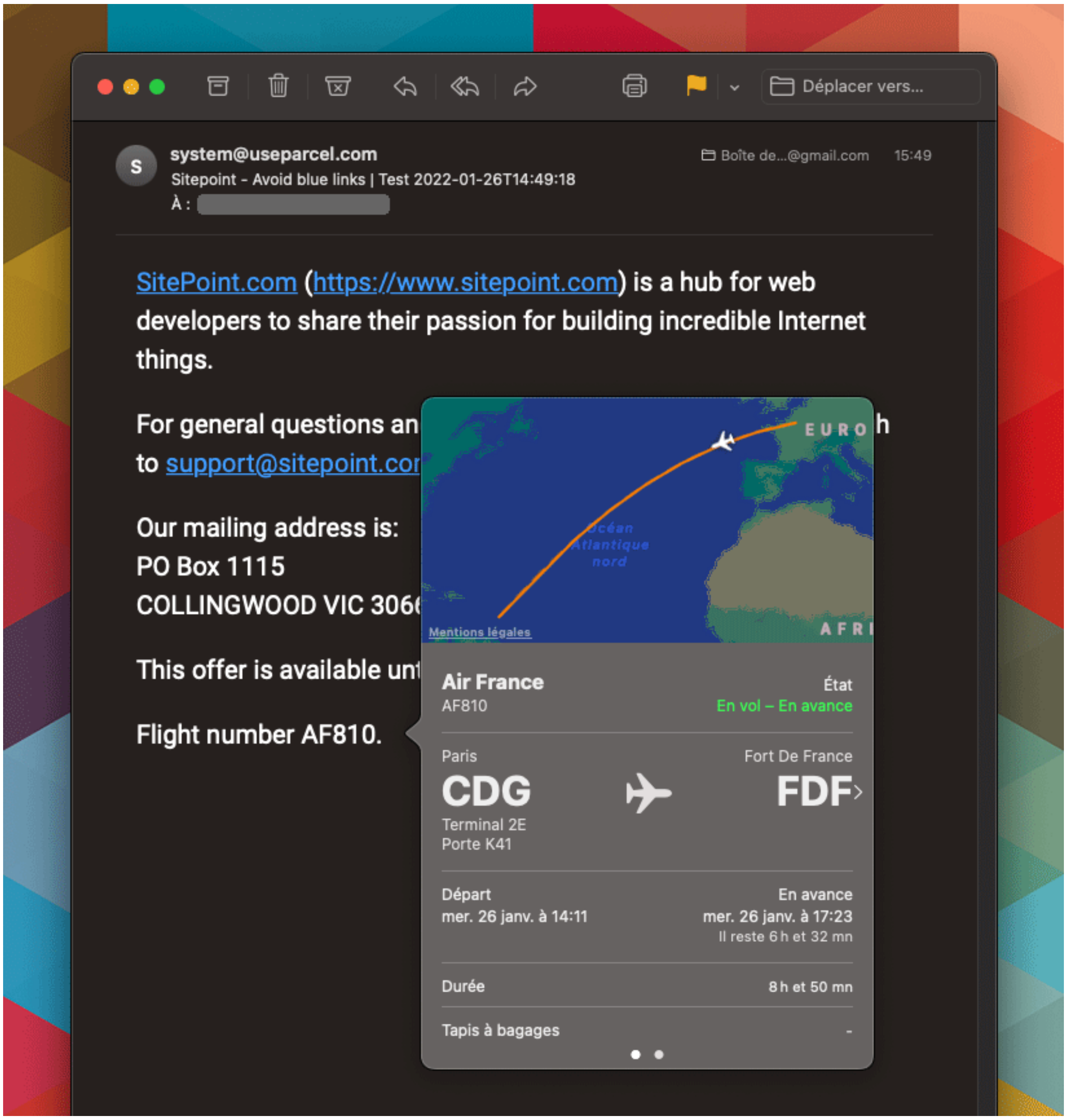
- **Layout**. An email without `<style>` should adjust to any width without horizontal scroll. I usually consider going as low as 280px wide, which reflects the width of an email viewed on Gmail on an iPhone SE.
- **Branding**. An email without `<style>` should reflect the branding of the sender.

Avoiding Automatic Links

Email clients automatically add links to certain keywords. This can happen to the following kinds of text:

- URLs (`sitepoint.com` , `https://www.sitepoint.com/blog/`)
- email addresses (`support@sitepoint.com`)
- phone numbers
- mailing addresses
- dates
- flight numbers

The image below shows how Apple Mail provides detailed information on air flights when a current flight number is clicked.



system@useparcel.com 15:49
Sitepoint - Avoid blue links | Test 2022-01-26T14:49:18
À : [redacted]

SitePoint.com (https://www.sitepoint.com) is a hub for web developers to share their passion for building incredible Internet things.

For general questions an h
to support@sitepoint.co

Our mailing address is:
PO Box 1115
COLLINGWOOD VIC 306

This offer is available un

Flight number AF810.

Air France État
AF810 En vol – En avance

Paris Fort De France
CDG FDF
Terminal 2E
Porte K41

Départ En avance
mer. 26 janv. à 14:11 mer. 26 janv. à 17:23
Il reste 6 h et 32 mn

Durée 8 h et 50 mn

Tapis à bagages -

2-1. Examples of automatically added links in Apple Mail

The problem is that, if we don't plan for them, these links will come up with their default styles (usually blue and underlined)—and this can create unexpected and undesirable effects. For example, an automatic blue link on an already blue background will be unreadable. Unfortunately, anticipating these automatic links isn't foolproof. It's not uncommon that a custom offer or tracking code is turned into a date or phone number link.

Here are three possible ways to avoid automatic links:

- **Add a specific meta declaration for Apple Mail**. With this tag in the `<head>` of our email, Apple Mail won't add automatic links:

  ```
   <meta name="format-detection" content="telephone=no, date=no, address=no,
  email=no, url=no">
  ```

- **Use a zero-width non joiner character**. A *zero-width non joiner* is an invisible character represented by the entity `&zwnj;` (or `&#847;`) in HTML. We'll use it here to break email clients' content detection algorithm by placing it in the middle of text where a link might be added:

  ```
   Visit Sitepoint&zwnj;.com for more details!
  ```

- **Add a link ourselves**. Email clients are clever enough to not try to add a link to text that's already linked. So if we add a link ourselves around text that we know might get automatically linked, we can apply our own styles. A link would still be there, but at least not so prominently visible:

  ```
   Visit <a href="https://sitepoint.com"
  style="color:#000; text-decoration:none;">sitepoint.com</a> for more details!
  ```

Using Real URLs

Microsoft's Outlook.com webmail has a bug when using non-URL text as the `href` value of an `<a>` element. For example, consider the following code:

```
<a href="" style="color:blue;">Get the app on iOS</a><br>
<a href="TBD" style="color:green;">Get the app on Android</a>
```

This is transformed by Outlook.com into the following:

```
Get the app on iOS<br>
[TBD] Get the app on Android
```

The entire `<a>` element, including its styles, is removed. And the value of the `href` attribute (if it's not empty) is added between brackets before the link text.

I've been caught by this bug more than once, usually because I didn't have the links to put there at the time I was coding. My go-to solution now is to always use the client's domain or `https://example.com` as a temporary link. Example.com is another handy domain that's reserved by IANA "for use in illustrative examples".

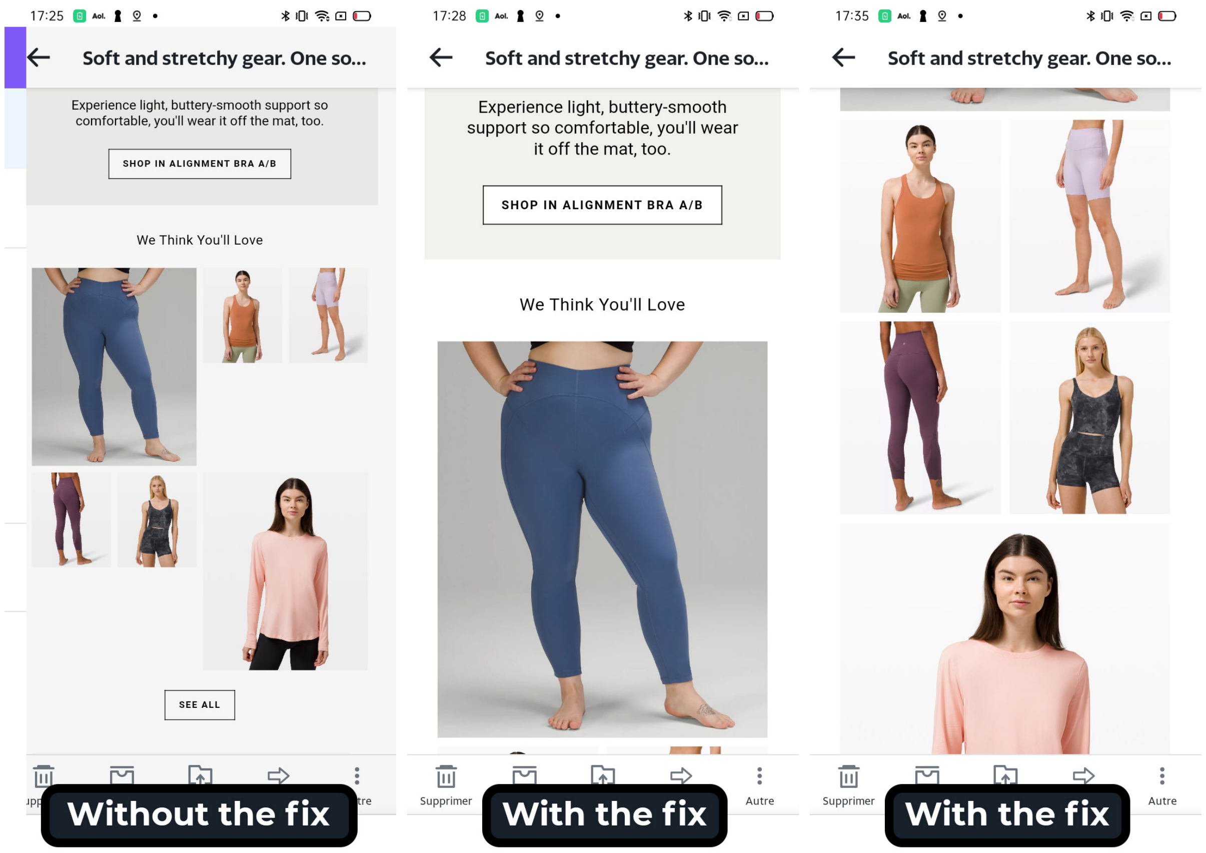
Adding an Empty `<head>`

Yahoo Mail on Android removes the `<head>` element of any email. This is very inconvenient, since this means that `<style>` tags, along with media queries, can't work there. But it was discovered that the Yahoo Mail app will only do this for the first `<head>` of an email.

So this means that if we include a dummy empty `<head>` first in our code, and then keep our regular `<head>` element, Yahoo Mail on Android will correctly keep it and interpret it:

```
<!DOCTYPE html>
<html lang="en"<html lang="en"><head>
    <title>My Email</title>
    <style><head>
        @media (max-width: 600px) {
            …
        }
    </style>
</head>
```

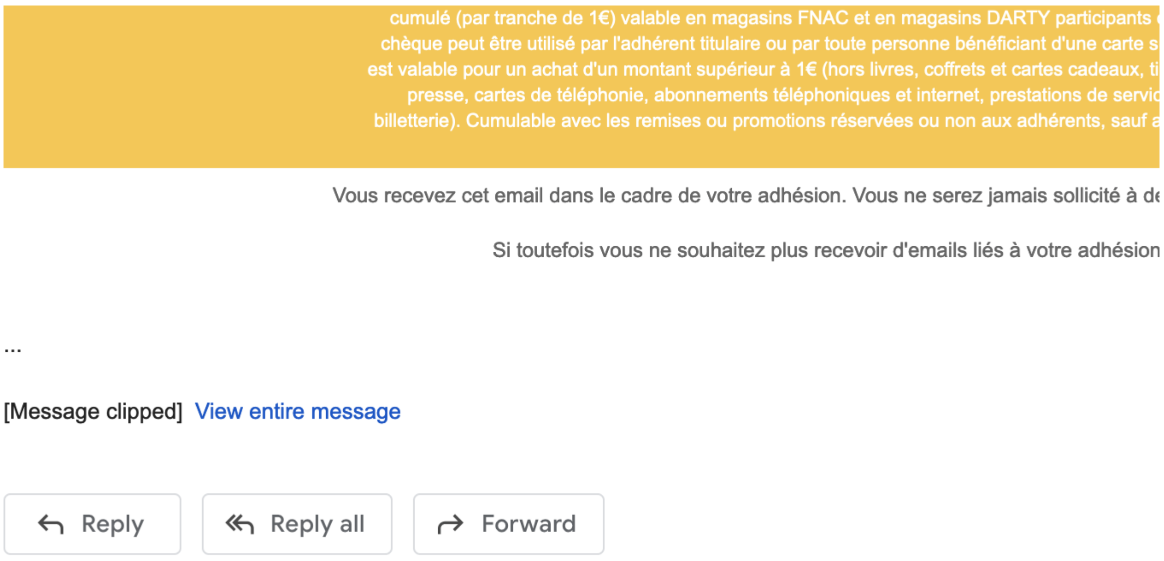
The image below shows an example email (courtesy of Really Good Emails) in the Yahoo Mail app on Android. Without the double `<head>`, the email is automatically scaled by the client. With the fix, the email renders media queries just as expected.


Soft and stretchy gear. One so...
Experience light, buttery-smooth support so comfortable, you'll wear it off the mat, too.
SHOP IN ALIGNMENT BRA A/B
We Think You'll Love
SEE ALL
Supprimer
Autre
Without the fix
With the fix
With the fix

2-2. Screenshots in Yahoo Mail Android showing the rendering with and without the double head hack

Keeping Email Sizes below `102KB`

Gmail's desktop webmail has a notorious threshold of 102KB, after which it will truncate an HTML email and show a "[Message clipped]" alert, along with a "View entire message" link. While it's unknown why this limitation exists and why it's at such an odd number precisely, it's been measured consistently across the years.


cumulé (par tranche de 1€) valable en magasins FNAC et en magasins DARTY participants
chèque peut être utilisé par l'adhérent titulaire ou par toute personne bénéficiant d'une carte s
est valable pour un achat d'un montant supérieur à 1€ (hors livres, coffrets et cartes cadeaux, ti
presse, cartes de téléphonie, abonnements téléphoniques et internet, prestations de servic
billetterie). Cumulable avec les remises ou promotions réservées ou non aux adhérents, sauf a

Vous recevez cet email dans le cadre de votre adhésion. Vous ne serez jamais sollicité à de

Si toutefois vous ne souhaitez plus recevoir d'emails liés à votre adhésion

...

[Message clipped] View entire message

2-3. Screenshot of a clipped email in Gmail

This 102KB limits only accounts **for the size of the HTML** of our email message. It doesn't include other parts of an email (like the plain text version, or all of the MIME header fields). It also doesn't include the weight of any distant asset like images or fonts. So if our HTML weighs 40KB with 500KB of images, we're fine.

We want to avoid this threshold because:

- Gmail will cut our email at 102KB to insert its "[Message clipped]" alert, even if this is right in the middle of a `<table>`. This is very likely to break our email in unexpected ways.
- Behind the "View entire message" link, Gmail will provide inferior support for HTML and CSS. For example, any `<style>` tag (and thus media queries) will be removed in this view.

A basic way to measure this is to look at the weight of our HTML file in our operating system. Online email tools like Alter.email or Parcel also show an estimate of our code weight.

Keep in mind that some email service providers may also process parts of our HTML in such a way that its weight will increase. For example, CSS inlining or link tracking can be big offenders in making our code overweight.

Optimizing our HTML email code can be done in several ways. Here's what I usually consider the most important steps:

- **Minify your code**. But don't use a minifier built for the Web! HTML Crush is a good one built for emails (avoiding making lines of code longer than 500 characters, for example). Just by removing indentations, you can diminish your HTML weight by a good third.

- **Remove unused code**. If your email is built from a generic template that includes lots of styles for different components, you might end up with styles you don't actually use. Online tool Email Comb cleans up unused code nicely.
- **Watch out for CSS inlining**. If you have a CSS rule targeting the universal selector (for example, `* { color:#000; }`), this will be applied to every single HTML element in your code (including `<tr>`, `<br>`, and plenty of other useless places). Make sure your CSS inliner works as you expect and inspect your code *after* inlining styles.
- **Use fewer tables**. Tables are necessary for Outlook on Windows, but they can be heavy. If we're not using `padding`, `border`, or multiple columns, it might be better and lighter to simply have more semantic bits of code stack on top of each other.

Gmail can also display the "[Message clipped]" alert without really clipping your message. This can be caused by the presence of special characters like `©` anywhere in our message.

Removing CSS Comments

Yahoo and AOL clients have a specific bug with CSS comments. For example, consider the following code:

```
<style>
    /* Big title */
    .title {
        font-size: 32px;
    }
</style>
```

This is transformed by Yahoo and AOL into the following:

```
<style>
    #yiv1234567890
    #yiv1234567890 .yiv1234567890title {
        font-size: 32px;
    }
</style>
```

The email client renames every class by adding a custom prefix (`yiv1234567890`) and adds a prefix to every selector (`#yiv1234567890`). This is a very common practice across email clients—especially webmail versions—and is needed to ensure that our email styles can't affect the webmail client's own styles, and vice versa.

The bug here is that Yahoo also tries to add its prefix to the CSS comment. And because it only adds it alone on a single line of code, this means it applies to the CSS selector on the next line. Consider the following selector:

```
#yiv1234567890 .yiv1234567890title { }
```

This now gets interpreted as follows:

```
#yiv1234567890 #yiv1234567890 .yiv1234567890title { }
```

With twice the `id` prefix, it no longer matches anything on the page, and thus this makes the CSS rule void.

This bug doesn't apply if we've got CSS comments inside a CSS rule (between the curly braces). But as a general cautionary rule, it's best to remove all CSS comments before sending an email.

Using an HTML5 Doctype

Email clients never output your HTML email code just as is. They apply various transformations, such as filtering unwanted tags (like `<script>`), attributes and styles. And webmail clients in particular don't keep your entire HTML.

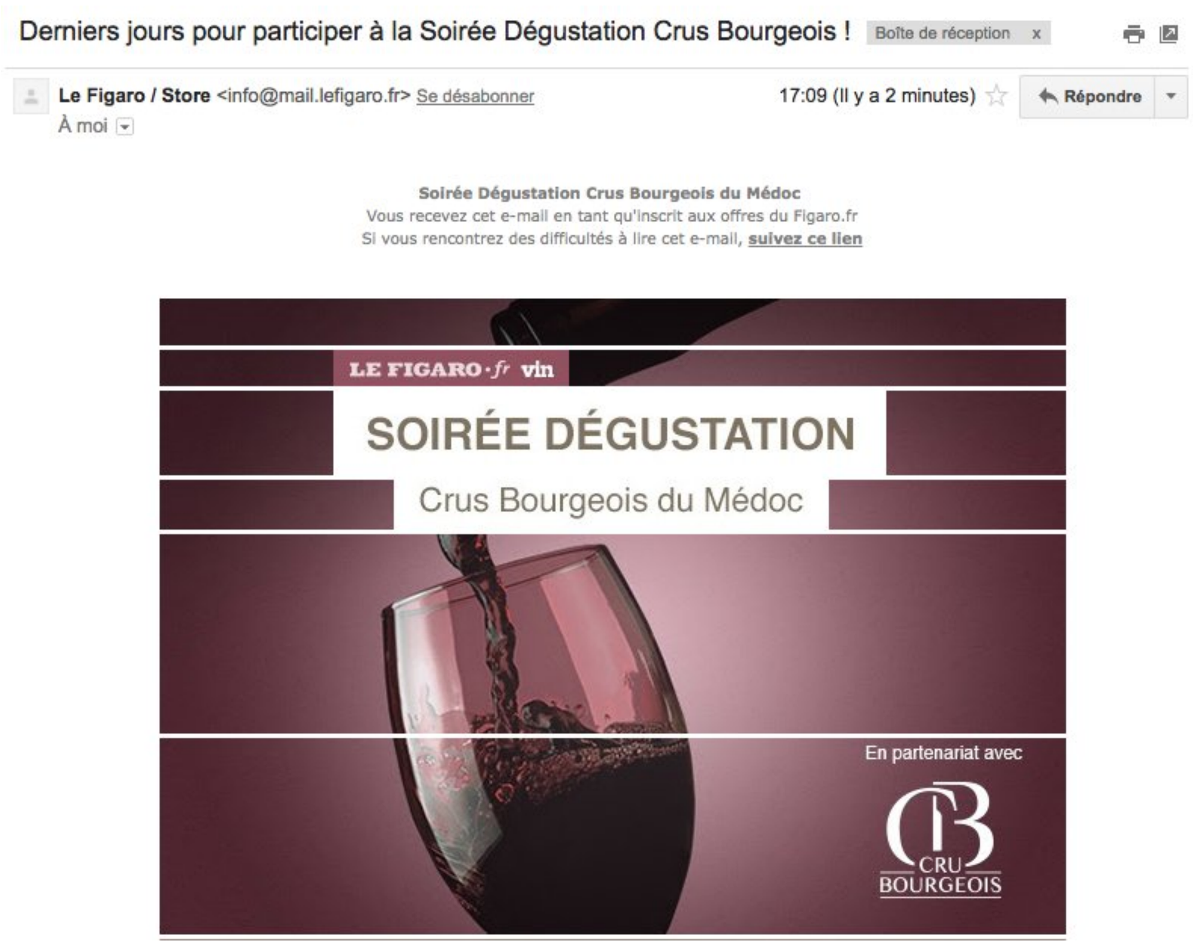
A webmail email is displayed in a web page that already has its own doctype, `<head>` and `<body>`, along with other meta elements. So a webmail client like Gmail looks for `<style>` tags in your head, compiles them into a single one, and keeps the content of your `<body>` element.

This means that, in a lot of cases, you'll end up with the webmail's doctype, not yours. And nowadays, most webmail clients use an HTML5 doctype:

```
<!DOCTYPE html>
```

One side effect of the HTML5 doctype is that `<img>` elements have a line spacing below them. This becomes clearly apparent when you slice images.

The following image demonstrates what happens when you forget to set `display:block` on your images in an HTML email (courtesy of @HTeuMeuleu on Twitter).


Derniers jours pour participer à la Soirée Dégustation Crus Bourgeois ! Boîte de réception x

Le Figaro / Store <info@mail.lefigaro.fr> Se désabonner 17:09 (Il y a 2 minutes) Répondre
À moi

Soirée Dégustation Crus Bourgeois du Médoc
Vous recevez cet e-mail en tant qu'inscrit aux offres du Figaro.fr
Si vous rencontrez des difficultés à lire cet e-mail, suivez ce lien

2-4. Screenshot of an email in Gmail with line spacing below images

There are various solutions for getting around this. Here are three, by order of personal preference:

- Add `vertical-align:middle` in an inline style on the `<img>` element. (This might impact surrounding text if you need to align the text and image differently.)
- Add `display:block` in an inline style on the `<img>` element. (This changes the flow of the content and might impact sibling elements.)
- Add `font-size:0` to the parent element of the `<img>`. (This might impact alternative text rendering.)

Here's an example of the first solution applied to an image with an inline `style` attribute:

```
<img src="logo.png"
     alt=""
```

```
    style="vertical-align:middle;" />
```

Conclusion

Dealing with email client quirks is part of the job of an email developer. It's a good idea to follow email developer communities to follow the latest updates and practices. I recommend the `#emailgeeks` hashtag on Twitter and the `#emailgeeks` Slack channel.

It can be tough to keep up to date with the new quirks and also move beyond the old ones. In 2015, I launched a GitHub repository to track email bugs, a collection of issues with active discussions and potential solutions for all the different email quirks and bugs. Over the years, it has gathered over a hundred issues, a quarter of which have since been fixed. This gives me good faith that by reporting what we see, email clients can improve and fix their own code. While it's a slow process, I do feel like HTML emails are moving towards a better future, with more interoperability and standards support.

And this is great news, because this can open the way for more fun and exciting features—such as interactivity! In the next tutorial in this series, we'll have a look at how we can make our emails more interactive with just CSS and HTML.

Chapter

Adding Interactivity to HTML Emails

3

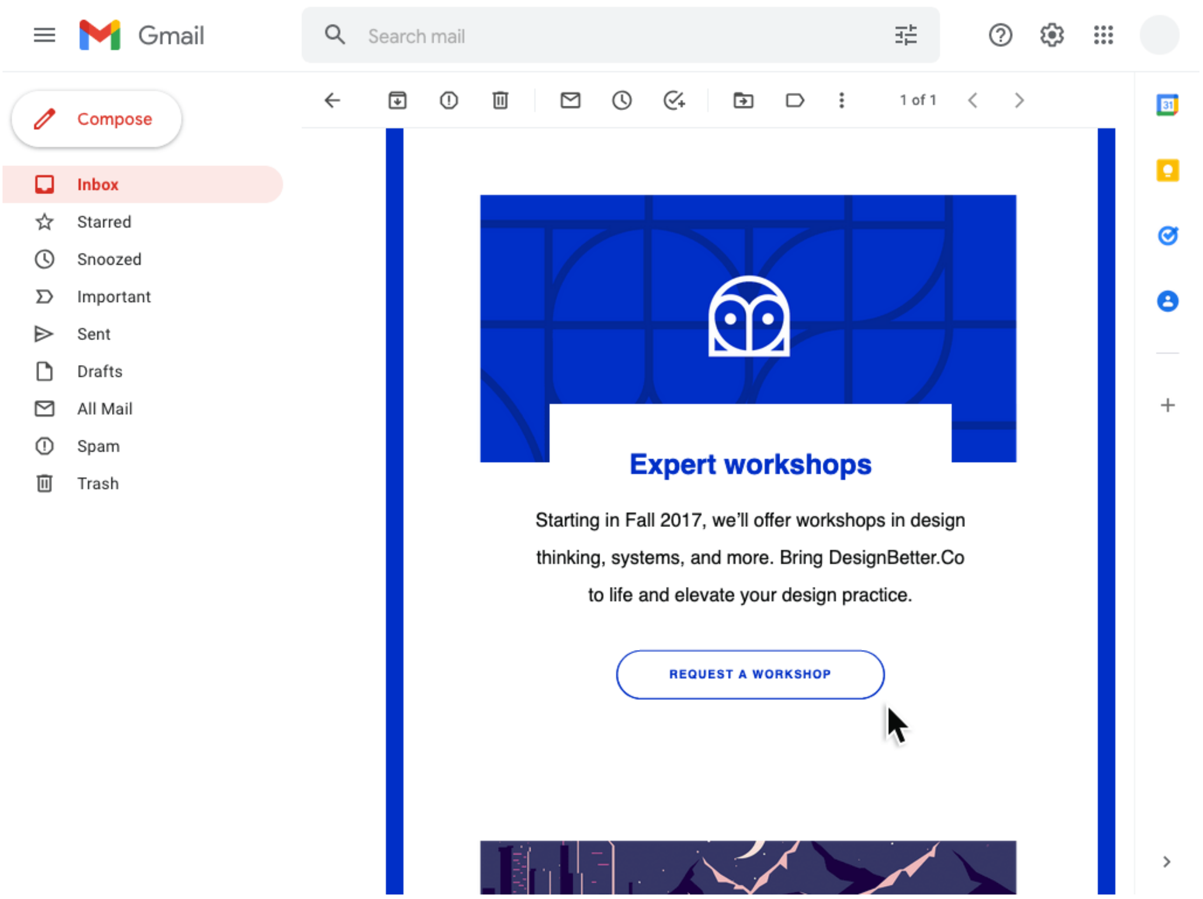
Rémi Parmentier

Interactive elements can greatly enhance HTML emails, making them stand out from the crowd. Even without JavaScript, we can use CSS pseudo-classes like `:hover` or `:checked` to create interactive components right inside HTML emails. And we can also do things like present buttons in a more engaging manner with hover effects, or add relevant content behind reveal buttons.

In this third tutorial of our HTML email series, we'll look at three examples of interactive content we can use in HTML emails today.

Hover Effects on Link Buttons

A hover effect on a link button is a simple interaction you can add to make your calls to action catchier and more attractive. The image below shows an example email from Really Good Emails with a hover link button effect, as seen in Gmail.



Expert workshops

Starting in Fall 2017, we'll offer workshops in design thinking, systems, and more. Bring DesignBetter.Co to life and elevate your design practice.

3-1. Animated screenshot showing a hover effect on a link button in Gmail

All we need for this is the CSS `:hover` pseudo-class. And support for this in email clients is actually pretty good. It works in Apple Mail (macOS, iOS), Gmail (desktop webmail), Outlook (macOS), Outlook.com, and Yahoo Mail (desktop webmail), among others.

There's only one quirk in Outlook.com where `:hover` (along with other pseudo-classes) is only supported on type selectors. So, for example, Outlook.com won't support `.button:hover`, but it will support `a:hover`. The usual fix for this is to add a `class` attribute to a parent element and use that element to target the link inside instead.

Here's an example of code for a link button in an HTML email:

```
<table class="email-btn" border="0" cellpadding="0" cellspacing="0" role="presentation"
  align="center"
  style="margin:0 auto; max-width:100%; background:#2ea44f; border-radius:5px;
  overflow:hidden; border-collapse:separate;">
    <tr>
        <td height="48" style="mso-padding-alt:0 16px; text-align:center;">
            <a href="https://www.example.com" target="_blank"
            style="display:block; min-width:128px; padding:14px 16px; color:#fff;
            font:bold 16px/20px Helvetica Neue, Roboto, sans-serif; text-decoration:none;
            text-align:center;">Call to Action</a>
        </td>
    </tr>
</table>
```

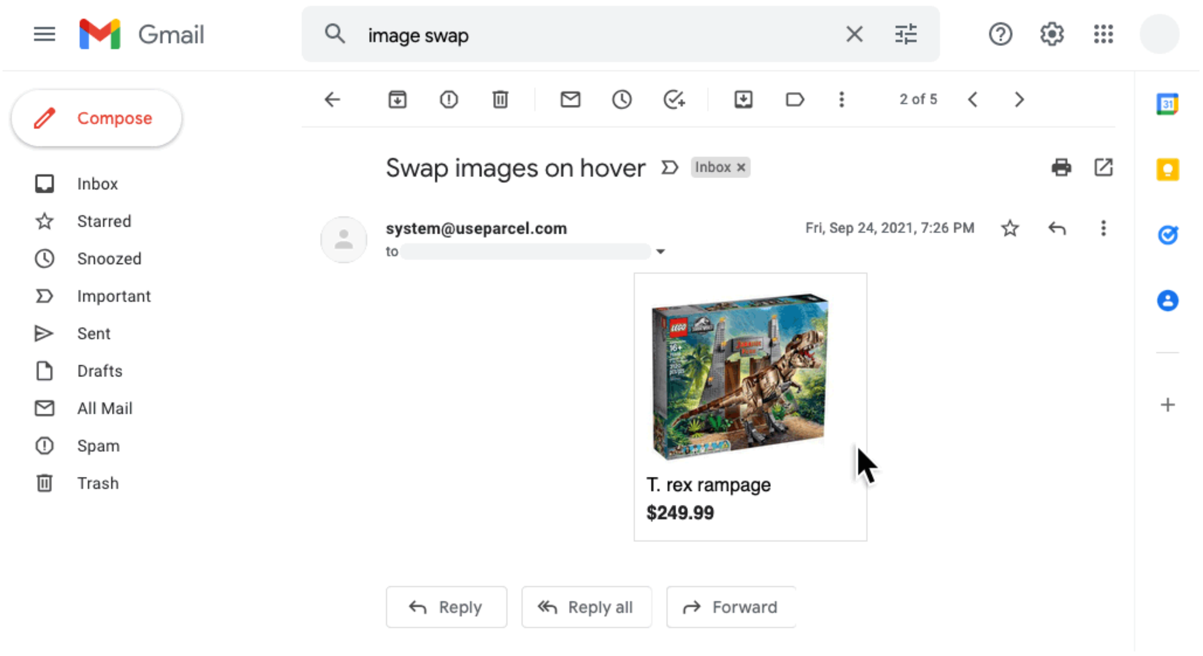
And here's the styling to get a simple color swap when hovering over this link button:

```
.email-btn a:hover {
    background-color: red !important;
}
```

According to Email Platform Status, only 0.02% of emails use `:hover`, while 66.67% of email clients tested in Can I email support it. So it's a great option for getting your email to stand out.

Swapping Images on Hover

Another interesting use case for `:hover` is to swap images. This is pretty common on the Web on ecommerce websites—for example, to show a different view of the same product, front by default, and back on hover.



3-2. Animated screenshot showing an image swap effect on hover in an email in Gmail

Here's the HTML for what's seen in the image above:

```
<div class="image-swap-group">
    <div>
        <img src="https://i.imgur.com/TSuCstim.jpg" alt="T. rex rampage box front view" width="160"
        height="160" style="vertical-align:middle;" class="image-front" />
        <!--[if !mso]><!-->
        <img src="https://i.imgur.com/aaCOaRUm.jpg" alt="T. rex rampage box back view" width="160"
        height="160" style="display:none; vertical-align:middle;" class="image-back" />
        <!--<![endif]-->
    </div>
</div>
```

And here's the CSS:

```
.image-swap-group div:hover .image-front {
    display:none !important;
}

.image-swap-group div:hover .image-back {
    display:block !important;
}
```

Here's how it works:

1 First, we have the two images, each with its own class (`image-front` and `image-back`).

2 Then we hide the second image by adding `display:none` in an inline style.

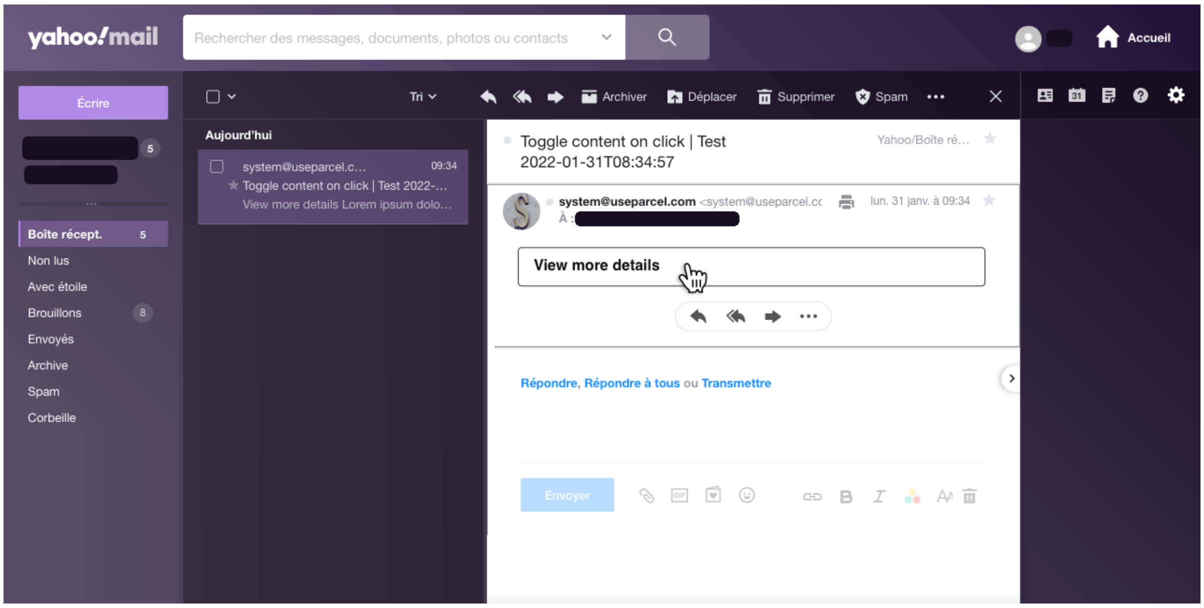
3 We also add conditional comments (`<!--[if !mso]><!-->`) around that second image to hide it from Outlook on Windows. (Another technique is to use the property `mso-hide:all` . I usually avoid this, because it doesn't work when applied directly to `<img>` elements, and it needs to be repeated on every `<table>` element you want to hide. It also breaks when replying or forwarding the email in Outlook, revealing the hidden content in plain sight.)

4 We wrap the two images in two `<div>` elements. The first one has a class of `image-swap-group` , which we're going to use to target the `<div>` inside. (This way, we only use a type selector, which is a workaround for the Outlook.com bug mentioned in the first button example.)

Toggling Content on Click

By combining HTML form controls—like checkboxes (`<input type="checkbox">`) or radio buttons (`<input type="radio">`)—and the CSS pseudo-class `:checked` , we can create more powerful interactions based on clicks (or taps, on touch devices). Let's see how we can create the following interface where a click on a button reveals new content.

The following image shows content being toggled on click in Yahoo Mail.



system@useparcel.c... 09:34
Toggle content on click | Test 2022-...
View more details Lorem ipsum dolo...

Toggle content on click | Test
2022-01-31T08:34:57

system@useparcel.com <system@useparcel.cc lun. 31 janv. à 09:34
À : [redacted]

View more details

Répondre, Répondre à tous ou Transmettre

3-3. Animated screenshot showing a content toggle on click in an email in Yahoo Mail

A Basic Checkbox

To get started, here's a basic checkbox HTML example:

```
<input type="checkbox" id="email-checkbox" />
<label for="email-checkbox">Toggle checkbox</label>
```

The `for` attribute is set with the value of the `id` from the checkbox. This links the two together so that it's not only better for accessibility, but it also allows for clicking on the `<label>` to toggle the checkbox value.

Using the CSS pseudo-class `:checked`, we can apply specific styles to the `<label>` when the checkbox is checked. For this, we're using the CSS adjacent sibling combinator (`+`), which allows for targeting an element (here, the label) directly next to another (the input) in the HTML:

```
input:checked + label {
    background-color: green;
}
```

Wrapping a Checkbox inside a Label

While the solution above works well in email clients like Apple Mail or Outlook.com, it doesn't work in Yahoo Mail. That's because, when parsing our HTML email code, Yahoo Mail changes `id` values by adding a random prefix. But it doesn't apply the same process to `for` attributes. Here's what our basic checkbox code from earlier looks like in Yahoo Mail:

```
<input type="checkbox" id="yiv0123456789email-checkbox" />
<label for="email-checkbox">Toggle checkbox</label>
```

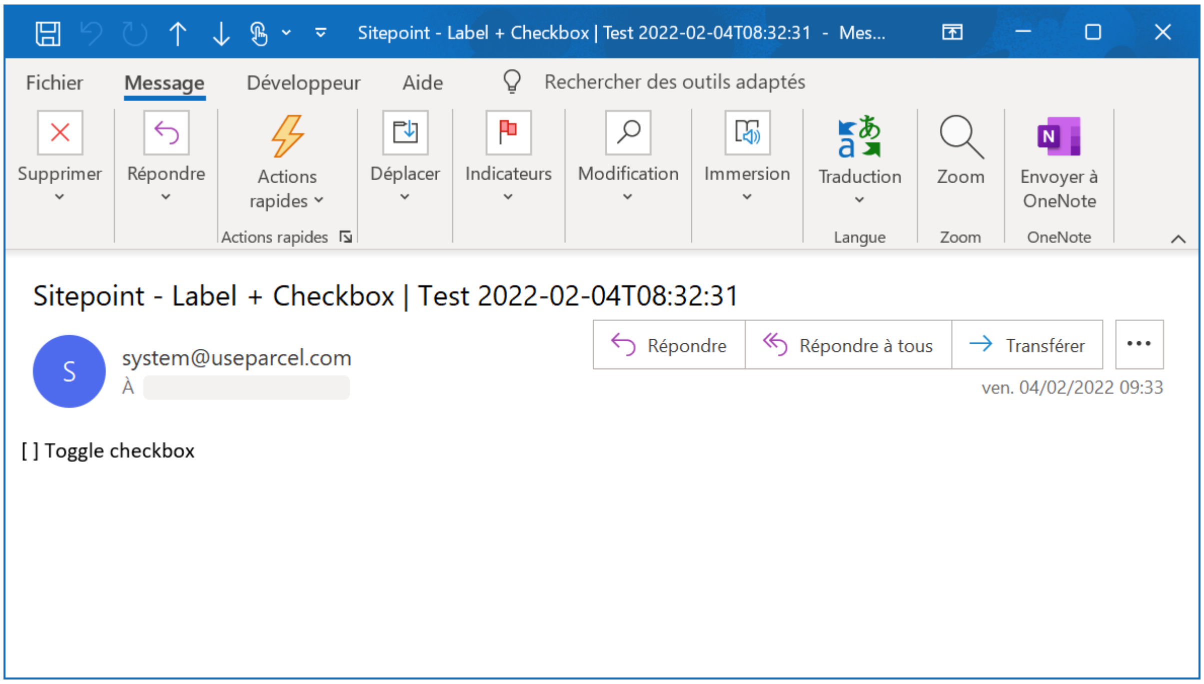
The `id` and `for` attribute values no longer match, meaning that any interactivity between the two is lost.

To get around this, we'll use another possible syntax where we nest the `<input>` element *inside* the `<label>`:

```
<label>
    <input type="checkbox" />
    <span>Toggle checkbox</span>
</label>
```

Conditional Comments for Outlook

As one might expect, Outlook on Windows doesn't support either of the form options above. It insists on transforming form elements into plain text. So our previous example would display as `[] Toggle checkbox` in Outlook on Windows, as pictured below.



Sitepoint - Label + Checkbox | Test 2022-02-04T08:32:31

system@useparcel.com

À

ven. 04/02/2022 09:33

[ ] Toggle checkbox

3-4. A checkbox showing as brackets in Outlook on Windows

To prevent this, I recommend wrapping the `<input>` element with `!mso` ("not mso") conditional comments to completely hide the checkbox from Outlook:

```
<label>
    <!--[if !mso]><!-->
    <input type="checkbox" />
    <!--<![endif]-->
    <span>Toggle checkbox</span>
</label>
```

Hiding the Checkbox in Other Email Clients

Since we don't want the checkbox to be visible, we'll hide it by default. `display:none` is a good starting point. (This makes our checkbox hidden for keyboard navigation or screen readers, which might be worth considering for accessibility. For a more accessible way to hide a checkbox, web developer Sara Soueidan has details on how to hide a checkbox inclusively.)

```
<label>
    <!--[if !mso]><!-->
    <input type="checkbox" style="display:none;" />
    <!--<![endif]-->
    <span>Toggle checkbox</span>
```

```
</label>
```

This works in most email clients. But there are a few email clients (like the French webmail clients of SFR and La Poste) that transform `<input>` elements into `<noinput>`. I presume they do this for security to parse out form elements. But `<noinput>` is not a real HTML element. And in HTML, if you use a made-up element like this, it will behave like a `<div>`. But because our `<noinput>` element doesn't have a matching closing tag, it will act as a wrapper for all the sibling content of our original checkbox. And because we have `display:none` on it, all this sibling content becomes hidden as well.

One way to get around this is to use use a `display:contents` declaration, after the previous `display:none`. If an email client supports `display:contents`, it will act as if the `<input>` (or `<noinput>`) element wasn't there in the first place. Otherwise, it will apply the `display:none` fallback and simply hide the checkbox:

```
<label>
    <!--[if !mso]><!-->
    <input type="checkbox" style="display:none; display:contents;" />
    <!--<![endif]-->
    <span>Toggle checkbox</span>
</label>
```

But wait! We're not done yet! That last addition now breaks our previously working code in the German webmail client T-Online. T-Online doesn't support `display:contents`. And it also removes any duplicate declaration for a CSS property, so our `display:none` fallback isn't there anymore. How do we get around this? We'll get our hands dirty and add a second `style` attribute, this time with only a `display:none` declaration:

```
<label>
    <!--[if !mso]><!-->
    <input type="checkbox" style="display:none; display:contents;" style="display:none;" />
    <!--<![endif]-->
    <span>Toggle checkbox</span>
</label>
```

A standard rendering engine would only keep the first occurrence of a duplicated HTML attribute. But T-Online's parser will keep the last one.

This is a great example of how HTML email coding is often a game of whack-a-mole, where fixing a bug in one client creates a new bug in another client. Sometimes, we need to use dirty hacks to go that extra mile and support a few more email clients.

Using the ~ General Sibling Combinator

In order to achieve our little interactive toggle component, we need one HTML element for the toggle button, and one for the actual content we want to hide or show. A basic version of our HTML would look like this:

```
<label class="email-toggle">
    <!--[if !mso]><!-->
    <input type="checkbox" style="display:none; display:contents;" style="display:none;" />
    <!--<![endif]-->
    <div class="email-toggle-button">
        View more details
    </div>
    <label class="email-toggle-content">
        Lorem ipsum dolor, sit amet consectetur adipisicing elit. […]
    </label>
</label>
```

Using a nested `<label>` for the `.email-toggle-content` element is a nice trick for allowing clicks inside the content without closing the accordion.

In order to target the `.email-toggle-content` content from the checkbox, we use the tilde (`~`) general sibling combinator in CSS:

```
.email-toggle input:checked ~ .email-toggle-content {
    display: block;
}
```

Since interactivity in email requires support for embedded styles (in a `<style>` element), we hide the button by default within an inline `style` attribute and revert back to showing it in a `<style>` element. And we can apply the same logic for the toggle content (letting it be visible by default and only hiding it through a `<style>` element):

```
<style data-embed>
.email-toggle-button {
    display: block !important;
}
.email-toggle-content {
    display: none;
}
</style>
```

The `data-embed` attribute on the `<style>` tag is there to make sure these styles don't get inlined when using a CSS inliner.

Here's a full working example on CodePen, complete with additional markup and styles for presentation.

Conclusion

I love email interactivity, because it opens up a whole new world of possibilities. The Really Good Emails website has a great gallery of interactive emails—from live hotspots (like in this email from the BBC), to carousels (like in this email from UGG), and even games like Minesweeper (by the email duo Camiah), or Super Mail Quest (by email developer Aaron Simmonds).

Coding interactive emails can be both challenging and fun. Email developer Mark Robbins—often considered the "godfather" of interactive emails—has great resources online:

- "Interactive Email", a conference talk at beyond tellerrand.
- "Build interactive emails with CSS", an article for net magazine.
- Good Email Code—Mark's website—where he shares detailed examples of common email code.

In the next tutorial, we'll move from interactivity to accessibility, and look at how we can improve our emails for people with disabilities.

Chapter

4

Accessibility in HTML Emails

Rémi Parmentier

Focusing on accessibility is a fundamental part of designing for the Web. It ensures that websites are designed and coded so that they can be used by people with disabilities. The same principles apply to emails. And I firmly believe that, as email developers, it's our job to make sure the emails we code are accessible to everyone. According to a yearly survey by WebAIM, some of the most common accessibility issues involve images and alternative text, headings and other HTML semantics, as well as language definitions.

In this tutorial, we'll look at three fundamental ways to make our email code more accessible.

Using Semantic Markup

HTML is a language. (It's right there in the name—HyperText Markup *Language*.) The best thing we can do as developers is to communicate in this language! But rather than HTML being a language spoken to other humans, it's primarily a language we use to communicate with other software (software which, of course, ends up being used by other humans). HTML is meant for browsers, search engine robots, and assistive technologies like screen readers, so that they can figure out the structure and hierarchy of our content. If we build a web page using only `<div>`s, these technologies will have a hard time figuring out what's important in our content, as will the humans using these technologies. Adopting semantic markup means using the right HTML element (or the right *word*) for the right content, thus making our emails more comprehensible.

Adding `role="presentation"` to Tables

Because of Outlook on Windows, we need to use `<table>` elements to lay out our emails. This is problematic, because a screen reader expects a `<table>` to contain tabular data. So it reads tables by enunciating each row and each cell. To stop it from doing this, we can use the `role="presentation"` attribute on every table used for layout:

```
<table role="presentation" border="0" cellpadding="0" cellspacing="0">
```

The `role="presentation"` attribute isn't inherited by nested tables (contrary to the `lang` attribute, for example). It's only inherited by a table's child elements (like `<tr>` or `<td>`). So it needs to be set on every presentational table. Some assistive technologies have heuristics for working out by themselves which tables are definitely there for layout. For example, Apple's VoiceOver on macOS is clever enough to guess that a table with only one line and one column is a layout table. But as a cautionary rule and a best practice, it's better to define this ourselves with the `role="presentation"` attribute on every `<table>` used for layout.

Here's a video of a screen reader interpreting an email *without* `role="presentation"`.

And here's a video of a screen reader interpreting the same email *with* `role="presentation"` added.

Using Headings, Paragraphs, and Lists

Another way to improve our email markup is by using adequate semantic HTML for different types of text content—such as headings (`<h1>` , `<h2>` , ..., `<h6>`), paragraphs (`<p>`), and lists (`<ul>` , `<ol>` , `<li>`).

A great advantage of such semantic elements is that a screen reader can provide additional information to users. For example, VoiceOver on macOS, when browsing a list, will announce the position of the element in the list.

Another gain from using semantic headings is that users can get a summary of the email they're reading and jump from one heading to another if they choose.

Adding `role="article"`

Using more structural semantic markup (like `<header>` , `<footer>` or `<main>`) is best practice on the Web. But not all email clients support these elements. For example, Gmail will replace `<main>` opening and closing tags with `<u></u>` , while Outlook.com will straight out remove the element and only leave its inner content.

Email developer Mark Robbins has been advocating for the use of a `role="article"` attribute on a wrapping `<div>` . This creates a landmark to make our email more easily accessible. Adding an `aria-label` attribute with our email subject line helps make this more specific and useful:

```
<div role="article" aria-label="Your email subject line">
```

The `alt` Attribute

The `alt` attribute gives screen reader users a sense of the content of an image. Let's look at some best practices for doing this in HTML emails.

Setting an Empty `alt` Attribute

Images must always have an `alt` attribute. Otherwise, a screen reader like VoiceOver will try to give its user clues about what the image might be about by reading the image file name. That can end up being okay if we have an image named `logo.png` , but it's likely to result in a pretty miserable user experience if our image name is automatically generated and looks like `a1b2-c3d4-e5f6-g7h8.jpg`.

Here's a video showing an example of VoiceOver on macOS reading out random image names.

By setting an `alt` attribute, even if it's empty, we're making sure that we're in control of the alternative text. We can set empty alternative text by specifying `alt=""` (with nothing between the quotes). A screen reader will then simply ignore that image as if it isn't there:

```
<img src="shadow.png" alt="" />
```

Any image that's purely decorative should have an empty `alt=""` attribute. This includes images such as a shadow effect, or a decorative icon or illustration. Be careful to not have any space in there, though. Using `alt=" "`, for example, could be picked by a screen reader as an image with a blank alternative text instead of no alternative text.

If an image is wrapped in a link, then having an empty `alt` attribute isn't a good option. Because the image is linked, a screen reader like VoiceOver will try to give its user a sense of what the link might be about. Unless there's also caption text within the link, `alt` text is an important means for informing users of what's being linked to. Otherwise, the screen reader will tend to read out image's file name or the link's URL. That can work out okay if the URL is something like `example.com/contact/`, but it's going to be a mess if the URL is automatically generated and looks something like `example.com/e5f6-g7h8-contact/`.

Here's a video showing an example of VoiceOver on macOS reading link URLs on a linked image.

Setting Appropriate Alternative Text

If an image is actually part of the content, then it must have adequate alternative text. WebAIM offers a set of guidelines for alternative text, including the following:

- Be **accurate and equivalent**. If our image contains some text, and that text is nowhere else on the page, then it should be used as the alternative text.
- Be **succinct**. Don't be overly descriptive. WebKit (Apple's rendering engine used on Apple Mail and across all iOS apps) hides alternative text if it doesn't fit in one line inside a blocked image's width. Also, it's not necessary to include words like "image of …". Assistive technologies like screen readers will already mention the fact that this is an image element.
- **Don't be redundant**. If our image is of a product and that product's name is right below the image, it's best to leave the alternative text empty.

The W3C Web Accessibility initiative offers an `alt` "decision tree" that can be very helpful if we're unsure what to use as alternative text.

The `lang` Attribute

Defining the language of the HTML content helps assistive technologies like screen readers pick the right voice for reading the content. The `lang` attribute needs to be defined with a valid language code on the opening `<html>` tag:

```
<html lang="en">
```

But because some email clients (especially webmail versions) remove the `<html>` element, the `lang` attribute also needs to be set on a wrapping element (for example, a `<div>`) within the `<body>`:

```
<!DOCTYPE html>
<html lang="en">
  <head></head>
  <body>
    <div lang="en">
    </div>
  </body>
</html>
```

Without the `lang` attribute, a screen reader will assume the content it's reading is in the same language as the default language the computer it is running on. I'm French, and my computer is set up in French. If I read an email written in English with VoiceOver on macOS, this video shows what I get *without* the `lang` attribute set.

And this video shows what I get *with* the `lang` attribute set.

A small attribute can make a huge difference. So remember to set languages correctly.

And if you have an international audience, the same logic applies for the `dir` attribute, which defines the direction of the text. For example, an email in Arabic will benefit from having `dir="rtl"` ("right to left") set, while an email in English (which might end up being read in an email client that reads right to left by default) can use the `dir="ltr"` ("left to right") attribute.

Conclusion

I love accessibility, because it's another rabbit hole of its own. Following best practices and guidelines for the Web (like the Web Content Accessibility Guidelines from the W3C) also works for HTML emails. But it's only a beginning, and not an end, as every person with disabilities will have their own needs.

To dive deeper into the subject, email developer Wilbert Heinen set up a huge list of resources about email accessibility, from articles to online presentations and tools.

For day-to-day work on emails, I like to use the Parcel accessibility checker or Accessible-Email.org, an online tool that can audit our HTML email code to improve accessibility. The latter looks for headings and landmarks, lets you hand-check your alternative text and link text, and also guides you on setting the document language and fixing layout tables.

In our next and final tutorial in this series, we'll see how to apply everything we've learned so far by working through a template redesign case study.

Chapter

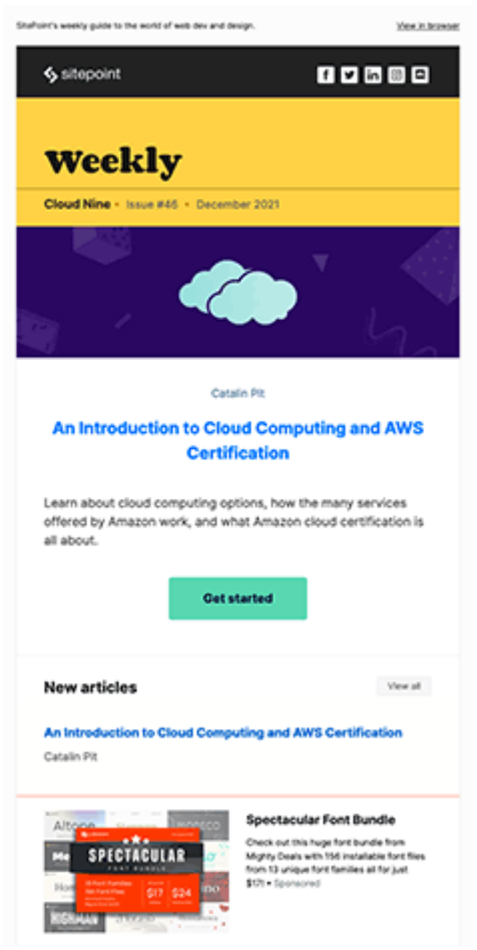
5

A Case Study: Redesigning SitePoint's Weekly Newsletter

Rémi Parmentier

In this final tutorial on crafting HTML emails, let's put everything we've learned into practice. I'm going to do an unsolicited redesign of SitePoint's Weekly Newsletter, focusing on improving the code of the HTML email.

The image below shows an example of the current SitePoint Weekly Newsletter layout.


SitePoint's weekly guide to the world of web dev and design.

View in browser

sitepoint

Weekly

Cloud Nine • Issue #46 • December 2021

Catalin Pit

An Introduction to Cloud Computing and AWS Certification

Learn about cloud computing options, how the many services offered by Amazon work, and what Amazon cloud certification is all about.

Get started

New articles

View all

An Introduction to Cloud Computing and AWS Certification

Catalin Pit

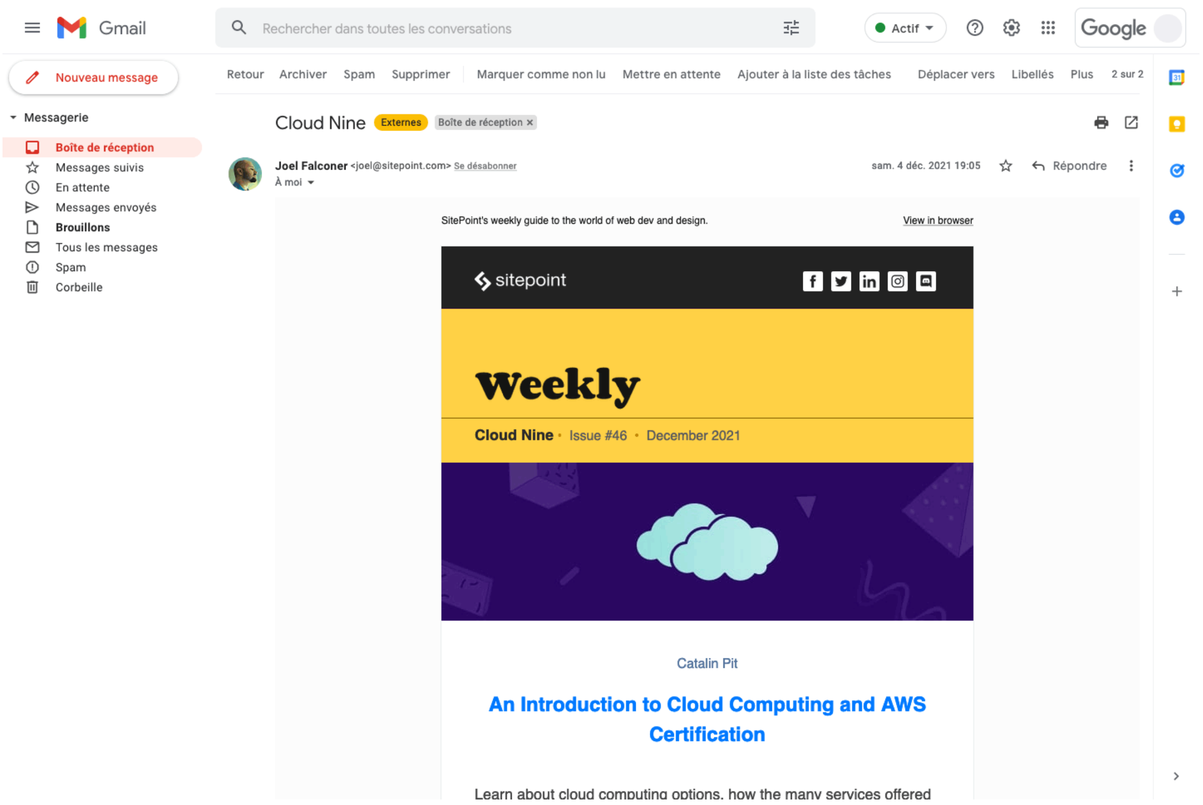
Spectacular Font Bundle

Check out this huge font bundle from Mighty Deals with 156 installable font files from 13 unique font families all for just $17! • Sponsored

5-1. A cropped screenshot of SitePoint Weekly Newsletter

A Full-width Header

Newsletters are often designed out of context. An email designed to be 600 pixels wide might look good in a Figma or Photoshop canvas. But what about when viewed inside a webmail client like Gmail on a large desktop screen?



Cloud Nine

Joel Falconer <joel@sitepoint.com> Se désabonner
À moi
sam. 4 déc. 2021 19:05

SitePoint's weekly guide to the world of web dev and design.

Weekly

Cloud Nine • Issue #46 • December 2021

Catalin Pit

An Introduction to Cloud Computing and AWS Certification

Learn about cloud computing options, how the many services offered

5-2. A screenshot of the SitePoint Weekly Newsletter viewed in Gmail desktop webmail with a narrow design

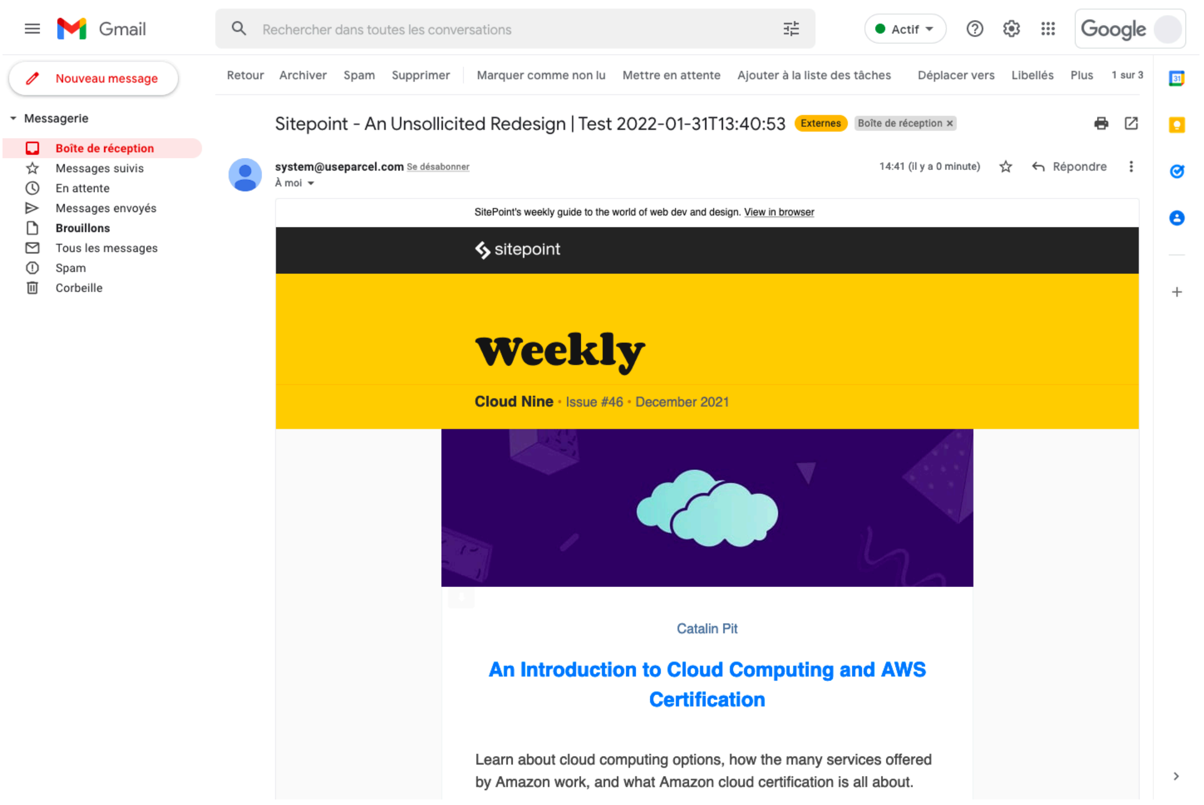
In my opinion, this kind of design looks a bit *floaty* and is begging to expand horizontally. One thing we can do is make the header and footer full width. In order to achieve this, we'll need two nested HTML containers: the outer one for the full-width background, and the inner one with a maximum width fixed.

Because of Outlook on Windows and Word's rendering engine, we'll use a `<table>` for the outer container, and a `<div>` with a `max-width:640px` for the inner one. But because Outlook on Windows doesn't support the `max-width` property on `<div>` elements, we're going to need to create one more table for Outlook with a fixed width of `640px`. But this time, it will be wrapped in conditional comments (`[if mso]`) and will only be visible for Outlook on Windows.

Here's the full code corresponding to such a full-width section:

```
<table style="width:100%; background-color:#ffd145;" border="0" cellpadding="0" cellspacing="0"
  role="presentation">
    <tr>
        <td>
            <!--[if mso]>
            <table style="width:640px;" border="0" cellpadding="0" cellspacing="0" align="center"
            role="presentation"><tr><td>
            <![endif]-->
            <div style="max-width:640px; width:100%; margin:0 auto;">
                …
            </div>
            <!--[if mso]>
            </td></tr></table>
            <![endif]-->
        </td>
    </tr>
</table>
```

And here’s the result in Gmail desktop webmail.



Sitepoint - An Unsollicited Redesign | Test 2022-01-31T13:40:53

system@useparcel.com

SitePoint's weekly guide to the world of web dev and design. View in browser

Cloud Nine • Issue #46 • December 2021

Catalin Pit

An Introduction to Cloud Computing and AWS Certification

Learn about cloud computing options, how the many services offered by Amazon work, and what Amazon cloud certification is all about.

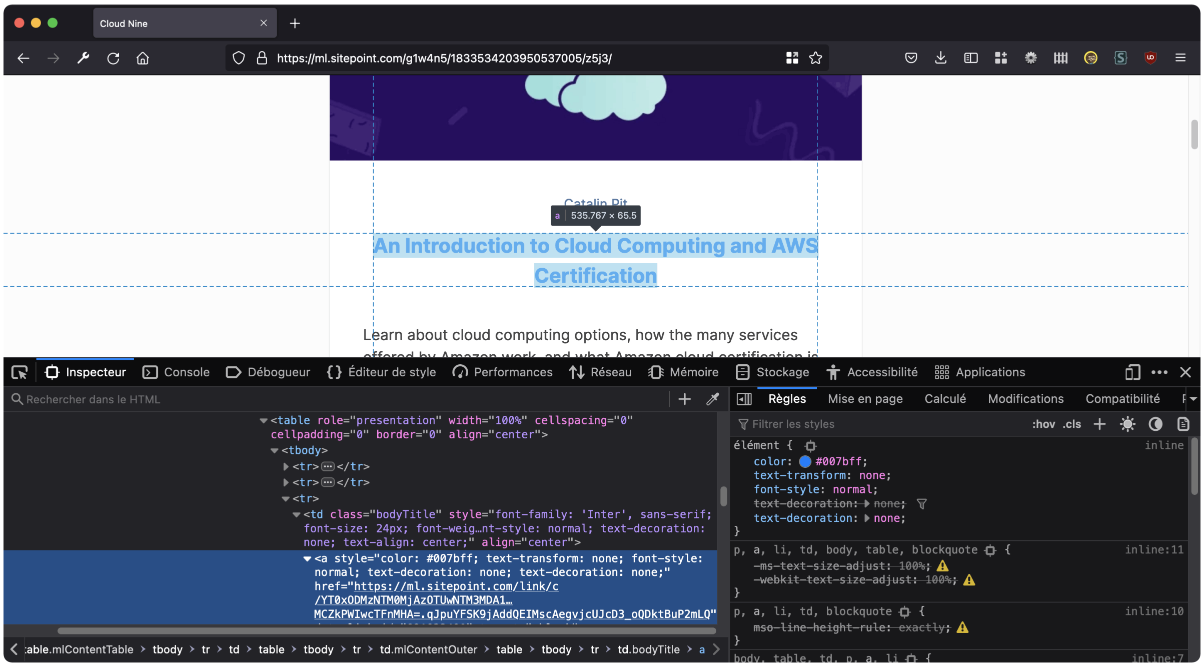
5-3. A screenshot of Sitepoint Weekly Newsletter viewed in Gmail desktop webmail, now with a full width header

This approach to coding email is often referred to as **fluid/hybrid**. Our email is coded to be fluid by default (using percentages and `max-width`). The beauty of it is that it works with just inline styles, with no `<head>` or media queries required. We *could* add additional styles in a media query if needed (thus making the email **hybrid**), but it would still be able to display properly without it. This is **progressive enhancement** in action.

One downside of this approach, however, is the reliance on HTML code inside conditional comments for Outlook on Windows. This kind of code is often referred to as **ghost tables**, because they're only visible to a select few (the Outlooks on Windows), and because they can be quite scary for newcomers. We need to test in Outlook on Windows to make sure we haven't broken anything. And if we need to change some design aspects (say, make the width `600px` instead of `640px`), we need to remember to change it in both the `<div>` *and* the `<table>` for Outlook.

A More Semantic Article Block

In SitePoint Weekly's original code, an article block is made up almost entirely of tables. Every block of text is a new table row, and even spacing is done with new empty rows. Not only is this bad for accessibility, it's a also a sure way to cross Gmail's dreaded 102KB weight limit.



```
<table role="presentation" width="100%" cellspacing="0"
cellpadding="0" border="0" align="center">
  <tbody>
    <tr>…</tr>
    <tr>…</tr>
    <tr>
      <td class="bodyTitle" style="font-family: 'Inter', sans-serif;
      font-size: 24px; font-weig…nt-style: normal; text-decoration:
      none; text-align: center;" align="center">
        <a style="color: #007bff; text-transform: none; font-style:
        normal; text-decoration: none; text-decoration: none;"
        href="https://ml.sitepoint.com/link/c
        /YT0xODMzNTM0MjAzOTUwNTM3MDA1…
        MCZkPWIwcTFnMHA=.qJpuYFSK9jAddQEIMscAegvjcUJcD3_oQDktBuP2mLQ"
```

```
:able.mlContentTable › tbody › tr › td › table › tbody › tr › td.mlContentOuter › table › tbody › tr › td.bodyTitle › a
```

```
élément {                                              inline
   color: #007bff;
   text-transform: none;
   font-style: normal;
   ~~text-decoration: none;~~
   text-decoration: none;
}
p, a, li, td, body, table, blockquote {           inline:11
   ~~-ms-text-size-adjust: 100%;~~
   ~~-webkit-text-size-adjust: 100%;~~
}
p, a, li, td, blockquote {                        inline:10
   ~~mso-line-height-rule: exactly;~~
}
body, table, td, p, a, li {                        inline:7
```

5-4. A screenshot of SitePoint's Weekly Newsletter code showing the use of tables

What I like to do to get a better sense of the structure of an HTML email is to start with just the bare bones. I'll first write down HTML code for the actual content. Since I've used an `<h1>` in the header for the newsletter title, I'm going to use an `<h2>` for every other block title in the newsletter. The rest of the text would be simple paragraphs:

```
<p>
    <img src="cloud.jpg" alt="" />
```

```
</p>
<p>
    Catalin Pit
</p>
<h2>
    <a href="…">An Introduction to Cloud Computing and AWS Certification</a>
</h2>
<p>
    Learn about cloud computing options […].
</p>
<p>
    <a href="…">Get started</a>
</p>
```

Now I can start adding specific styles. I'll start by setting the expected margin between each block of text. And instead of using empty table rows, I'm going to use the CSS `margin` property:

```
<p style="margin:0;">
    <img src="cloud.jpg" alt="" />
</p>
<p style="margin:0 0 20px;">
    Catalin Pit
</p>
<h2 style="margin:0 0 40px;">
    <a href="…">An Introduction to Cloud Computing and AWS Certification</a>
</h2>
<p style="margin:0 0 40px;">
    Learn about cloud computing options […].
</p>
<p style="margin:0;">
    <a href="…">Get started</a>
</p>
```

In order to create extra spacing around the text (but not around the image), I'm going to group all the text elements together. And once again, because Outlook on Windows doesn't support `padding` on all elements, I'm going to use a table:

```
<p style="margin:0;">
    <img src="cloud.jpg" alt="" />
</p>
<table style="width:100%; background-color:#fff;" border="0" cellpadding="0" cellspacing="0"
role="presentation">
    <tr>
        <td style="padding:40px 40px 50px;" class="sitepoint-mobile-padding-h-15">
            <p style="margin:0 0 20px;">
                Catalin Pit
            </p>
            <h2 style="margin:0 0 40px;">
                <a href="…">An Introduction to Cloud Computing and AWS Certification</a>
            </h2>
            <p style="margin:0 0 40px;">
                Learn about cloud computing options […].
            </p>
            <p style="margin:0;">
                <a href="…">Get started</a>
            </p>
        </td>
    </tr>
</table>
```

Responsive versus Mobile First

One thing I'm doing here is setting the `padding` for the desktop view and using a class (`.sitepoint-mobile-padding-h-15`) to change the value for mobile devices in a media query:

```
@media only screen and (max-width: 640px) {
    .sitepoint-mobile-padding-h-15 {
        padding-left: 15px !important;
        padding-right: 15px !important;
    }
}
```

This is typically what email developers refer to as **responsive web design**: focusing on the desktop first, and mobile second. The biggest downside is that, for email clients that don't support media queries or `<style>` elements, the `40px` horizontal desktop padding here would be pretty big.

Another approach is to code the email with a **mobile-first** mindset. Set the padding for its mobile value first (`15px`), and then use a media query to enhance it on desktop to a larger value (`40px`). When coding mobile first, we need to consider the rendering in Outlook on Windows. Here, we use a different proprietary property (`mso-padding-alt`) that will only apply in Outlook.

Here's the HTML:

```
<td style="padding:40px 15px 50px; mso-padding-alt:40px 40px 50px;"
  class="sitepoint-desktop-padding-h-40">
```

And here's the CSS:

```
@media only screen and (min-width: 640px) {
    .sitepoint-desktop-padding-h-40 {
        padding-left: 40px !important;
        padding-right: 40px !important;
    }
}
```

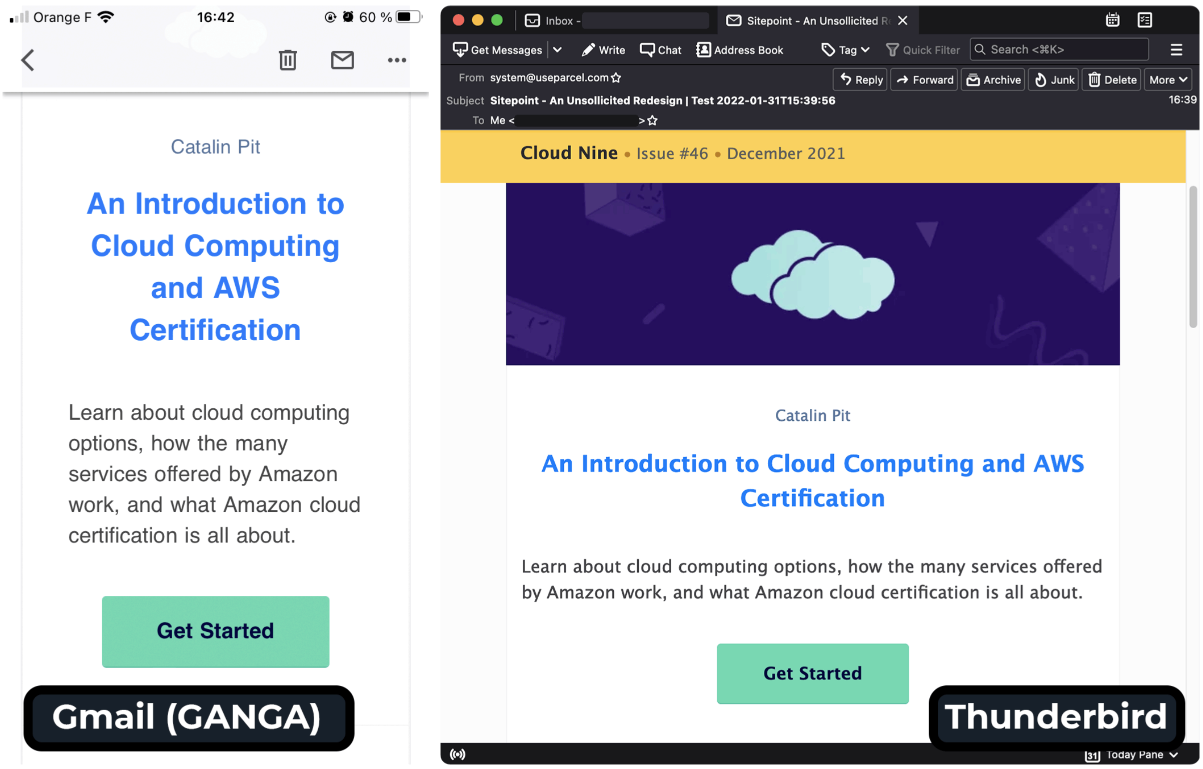
The difference between responsive, fluid/hybrid and mobile-first layouts is subtle nowadays. And in my opinion, what matters is not how well the layout makes your email *work*, but how well it makes your email *break*.

To show the difference between these techniques, you need to look for edge-case contexts, like these:

- A mobile device with no media query support—for example, in the Gmail apps with non-Gmail accounts, also known as GANGA. The responsive approach will fail here and show a desktop version of the styles, which might not work well.
- A desktop client with no media query support—for example, in Mozilla Thunderbird. The mobile-first approach will fail here and show the mobile version of the styles.
- A desktop webmail client with a small viewport. Even on a screen offering a 1440 wide pixel resolution, for example, the default configuration of Outlook.com only offers a width of 450 pixels for the email display. Both the responsive and mobile-first approaches might fail here and show a desktop version of the email, while we'd probably want something more mobile-like.

There's no universal answer to which approach is better here. It depends on the context, on your audience, on their email clients, and on your own coding abilities.

The image below compares two layouts. The left side shows Gmail on iOS, rendering an email from a non-Gmail account, without media query support, using a responsive layout with desktop padding. The right side shows a layout in Thunderbird on macOS, which doesn't offer media query support.



From system@useparcel.com
Subject Sitepoint - An Unsollicited Redesign | Test 2022-01-31T15:39:56
To Me < [redacted] >
16:39

Cloud Nine • Issue #46 • December 2021

Catalin Pit

An Introduction to Cloud Computing and AWS Certification

Learn about cloud computing options, how the many services offered by Amazon work, and what Amazon cloud certification is all about.

Get Started

5-5. Comparison screenshots of rendering a responsive email or a mobile first email

Conclusion

What I like about all this is that it makes the email developer's job very nuanced. We're the ones making choices that will inevitably impact rendering and support. And in the end, it makes email development a very human job.

This Pen shows the final version of this unsolicited redesign and recode.

Our little journey into HTML emails has come to an end. I hope I've successfully demonstrated that emails are still alive and well, and certainly not stuck in the 1990s.

When you understand the quirks of the most important email clients—especially Outlook on Windows—and the hacks for getting around them, you're in a good position to make your emails both more robust and more modern—utilizing many HTML and CSS features available to the best email clients. You can add interactive components, and you can even make your code more accessible.

HTML email has seen a lot of new features become available over the last decade—from responsive layout to dark mode. And there's good reason to get excited about what might be just around the corner for HTML email!