

MANUEL PRATIQUE

MJML vers JSON

Conversion, validation et rendu HTML sur Mac et en ligne

MJML — Mailjet Markup Language

JSON — JavaScript Object Notation

HTML — HyperText Markup Language

Version 001 — 8 octobre 2026

Nom canonique du fichier :

MANUEL_MJML_VERS_JSON_FR_2026-10-08_V001.docx

1. Objet du manuel

Ce manuel explique comment transformer un document MJML — Mailjet Markup Language — en un objet JSON — JavaScript Object Notation — exploitable par un programme, puis comment vérifier ce JSON, le modifier et produire à nouveau un email HTML — HyperText Markup Language. Il est conçu pour un utilisateur Mac qui souhaite comprendre le mécanisme, pas seulement exécuter une commande.

Principe fondamental. MJML et JSON ne sont pas deux rendus visuels différents : ce sont deux représentations possibles de la même arborescence logique d'un email.

2. Ce qu'est réellement la conversion MJML → JSON

Un fichier MJML est écrit avec des balises, par exemple `<mj-section>`, `<mj-column>` et `<mj-text>`. La conversion en JSON consiste à transformer cette hiérarchie de balises en objets possédant des propriétés. La documentation officielle MJML définit un composant JSON avec trois éléments : un `tagName` sous forme de chaîne, un objet `attributes`, puis soit une chaîne `content`, soit un tableau `children`.

MJML	JSON	Rôle
<code><mj-text></code>	<code>"tagName": "mj-text"</code>	Nom du composant
<code>color="#D02028"</code>	<code>"attributes": {"color": "#D02028"}</code>	Attributs du composant
Visages pour le Futur	<code>"content": "Visages pour le Futur"</code>	Contenu textuel
balises imbriquées	<code>"children": [...]</code>	Sous-composants

3. Exemple minimal : avant et après

3.1 MJML — Mailjet Markup Language source

```
<mjml>
  <mj-body>
    <mj-section>
      <mj-column>
        <mj-text color="#D02028">
          Visages pour le Futur
        </mj-text>
      </mj-column>
    </mj-section>
  </mj-body>
</mjml>
```

3.2 JSON — JavaScript Object Notation correspondant

```
{
  "tagName": "mjml",
```

```

"attributes": {},
"children": [
  {
    "tagName": "mj-body",
    "attributes": {},
    "children": [
      {
        "tagName": "mj-section",
        "attributes": {},
        "children": [
          {
            "tagName": "mj-column",
            "attributes": {},
            "children": [
              {
                "tagName": "mj-text",
                "attributes": {
                  "color": "#D02028"
                },
                "content": "Visages pour le Futur"
              }
            ]
          }
        ]
      }
    ]
  }
]
}

```

À retenir. Le JSON conserve la structure logique de l'email. Il ne contient pas encore le HTML final produit pour Gmail, Apple Mail ou Outlook.

4. Méthode recommandée en 2026 : mjml-parser-xml

Pour une conversion automatisée actuelle, la méthode recommandée dans ce manuel utilise le paquet `mjml-parser-xml` de l'écosystème MJML. Il transforme la syntaxe XML du document MJML en objet JavaScript. L'objet est ensuite sérialisé avec `JSON.stringify` pour créer un fichier `.json`.

Cette approche est préférable à l'ancien paquet `mjml2json` pour un nouveau workflow, car `mjml-parser-xml` fait partie des dépendances actuelles de l'outil de ligne de commande MJML et suit le développement récent de l'écosystème.

4.1 Préparer le Mac

Il faut disposer de Node.js et de NPM — Node Package Manager.

1. Installer Node.js depuis le site officiel Node.js, ou vérifier sa présence dans Terminal.

```
node --version  
npm --version
```

Si les deux commandes affichent un numéro de version, l'environnement est prêt.

4.2 Créer un dossier de travail

```
mkdir -p ~/Desktop/MJML_JSON  
cd ~/Desktop/MJML_JSON
```

4.3 Installer le parseur

```
npm init -y  
npm install mjml-parser-xml
```

4.4 Créer le script de conversion

Créer un fichier nommé `convert-mjml-json.mjs` contenant :

```
import fs from "node:fs";  
import parse from "mjml-parser-xml";  
  
const inputPath = process.argv[2];  
const outputPath = process.argv[3] || inputPath.replace(/\.mjml$/i, ".json");  
  
if (!inputPath) {  
  console.error("Usage : node convert-mjml-json.mjs fichier.mjml [fichier.json]");  
  process.exit(1);  
}  
  
const mjml = fs.readFileSync(inputPath, "utf8");  
const tree = parse(mjml);  
  
fs.writeFileSync(  
  outputPath,  
  JSON.stringify(tree, null, 2),  
  "utf8"  
);  
  
console.log(`JSON créé : ${outputPath}`);
```

4.5 Convertir un fichier

```
node convert-mjml-json.mjs DE_EPS_01.mjml DE_EPS_01.json
```

Le fichier DE_EPS_01.json contient alors l'arborescence MJML sous forme d'objets JSON lisibles et modifiables.

5. Méthode en ligne : sans installation locale

La conversion peut être effectuée entièrement dans le navigateur avec un environnement Node.js en ligne, par exemple StackBlitz. Le principe est identique : créer un projet Node.js, installer mjml-parser-xml, ajouter le fichier .mjml et exécuter le script de conversion.

2. Créer un projet Node.js dans StackBlitz.
3. Ajouter le fichier .mjml au projet.
4. Dans le terminal du projet, exécuter : `npm install mjml-parser-xml`.
5. Créer `convert-mjml-json.mjs` avec le script du chapitre 4.
6. Exécuter le script puis récupérer le fichier .json généré.

Confidentialité. Pour un document non public, l'exécution locale sur le Mac est préférable : aucun contenu n'a besoin d'être envoyé à un service tiers.

6. Ancienne méthode : paquet mjml2json

Il existe un paquet historique nommé `mjml2json`, créé précisément pour convertir une syntaxe MJML XML en JSON. Il reste utile pour comprendre le workflow ou pour des projets anciens, mais il n'est pas le choix privilégié pour un nouveau système : sa publication NPM est ancienne alors que l'écosystème MJML principal a continué d'évoluer.

```
npm install mjml2json
```

Dans un projet existant qui utilise déjà ce paquet et fonctionne correctement, il n'est pas nécessaire de le remplacer sans raison. Pour un nouveau projet, préférer `mjml-parser-xml`.

7. Vérifier que le JSON est correct

Un JSON syntaxiquement valide n'est pas nécessairement un JSON MJML valide. La vérification doit donc se faire à deux niveaux.

7.1 Vérification JSON

- Le fichier doit commencer par un objet `{ ... }`.
- Les noms de propriétés et les chaînes doivent utiliser des guillemets doubles dans le fichier JSON.
- Aucune virgule finale ne doit rester après le dernier élément d'un objet ou d'un tableau.
- La structure doit pouvoir être chargée par `JSON.parse` sans erreur.

```
node -e 'JSON.parse(require("fs").readFileSync("DE_EPS_01.json","utf8")); console.log("JSON OK")'
```

7.2 Vérification MJML

- La racine doit normalement être `"tagName": "mjml"`.
- Un document d'email contient généralement un enfant `"mj-body"`.
- Les composants doivent posséder `"attributes": {}` même lorsque cet objet est vide.
- Un composant terminal comme `"mj-text"` utilise généralement `"content"`.
- Un composant contenant d'autres composants utilise `"children"`.

8. Visualiser le JSON

Un visualiseur JSON affiche la structure, mais pas l'email. Il peut présenter l'objet sous forme de texte indenté, d'arbre repliable ou de graphe. Cette étape sert à comprendre et modifier les données.

Outil	Ce qu'il montre	Ce qu'il ne montre pas
Éditeur texte	JSON brut	Rendu de l'email
JSON viewer	Arbre / objets / tableaux	Rendu de l'email
MJML + <code>mjml2html</code>	HTML final	Structure JSON comme arbre

9. Transformer le JSON MJML en HTML

La documentation officielle MJML indique qu'une définition JSON peut être transmise directement à la fonction `mjml2html`. Il n'est donc pas obligatoire de reconstruire un fichier `.mjml` textuel avant de produire l'HTML.

9.1 Installer MJML

```
npm install mjml
```

9.2 Script JSON → HTML

```
import fs from "node:fs";
import mjml2html from "mjml";

const json = JSON.parse(
  fs.readFileSync("DE_EPS_01.json", "utf8")
);

const result = await mjml2html(json, {
  validationLevel: "soft"
});

if (result.errors?.length) {
  console.error(result.errors);
}

fs.writeFileSync(
  "DE_EPS_01.html",
  result.html,
  "utf8"
);

console.log("HTML créé : DE_EPS_01.html");
```

Enregistrer ce script sous `json-to-html.mjs` puis exécuter :

```
node json-to-html.mjs
```

Le fichier `DE_EPS_01.html` peut ensuite être ouvert directement dans Safari, Firefox ou Chrome.

10. Validation stricte

Pour détecter plus tôt les erreurs de composants ou d'attributs, utiliser le niveau de validation strict. La documentation officielle décrit trois niveaux : `strict`, `soft` et `skip`.

```
const result = await mjml2html(json, {
  validationLevel: "strict"
});
```

Recommandation. Utiliser `soft` pendant la construction et `strict` avant archivage ou production. Le mode `skip` doit rester exceptionnel.

11. Modifier le JSON automatiquement

L'intérêt principal du JSON est la manipulation programmatique. On peut chercher un composant, modifier un texte, changer une couleur, remplacer une image ou générer plusieurs variantes sans reconstruire manuellement toutes les balises.

11.1 Exemple : modifier un texte

```
function walk(node) {
  if (!node || typeof node !== "object") return;

  if (node.tagName === "mj-text" &&
    node.content?.includes("Visages pour le Futur")) {
    node.content = "Visages pour le Futur — édition 2026";
  }

  for (const child of node.children || []) {
    walk(child);
  }
}

walk(json);
```

11.2 Exemple : remplacer une couleur

```
function replaceColor(node, from, to) {
  if (!node || typeof node !== "object") return;

  for (const [key, value] of Object.entries(node.attributes || {})) {
    if (value === from) {
      node.attributes[key] = to;
    }
  }

  for (const child of node.children || []) {
    replaceColor(child, from, to);
  }
}

replaceColor(json, "#D02028", "#B51F28");
```

12. Conversion inverse : JSON → MJML

Si l'objectif final est uniquement de voir ou d'envoyer l'email, la conversion JSON → HTML avec `mjml2html` est suffisante. Si l'on veut retrouver un fichier `.mjml` éditable, il faut sérialiser l'arborescence JSON en syntaxe XML MJML.

Un paquet historique nommé `json2mjml` existe pour cette opération, mais il est ancien. Pour un workflow durable, il faut tester sa compatibilité avec la version MJML utilisée avant de l'intégrer à une chaîne de production.

Point important. JSON → HTML est pris en charge directement par MJML. JSON → texte MJML est une étape supplémentaire qui n'est pas nécessaire pour le rendu.

13. Cas particuliers à surveiller

13.1 `mj-include`

Le composant `mj-include` peut dépendre de fichiers externes. Une conversion limitée au texte principal peut donc perdre la relation avec ces fichiers si le chemin de base et les chemins autorisés ne sont pas gérés. Conserver le dossier source et documenter les dépendances.

13.2 Images

Une image référencée par un chemin local, par exemple `assets/blason_thonex.png`, peut être visible lors d'un test local, mais un email réellement envoyé nécessite généralement une ressource accessible avec une adresse HTTPS — HyperText Transfer Protocol Secure.

13.3 Contenu HTML dans `mj-text`

Le contenu d'un composant `mj-text` peut contenir du HTML interne. Il doit être traité comme du contenu, et non comme un nouveau niveau `children`, sauf si le parseur le représente explicitement autrement.

13.4 Composants personnalisés

Des composants communautaires ou personnalisés peuvent nécessiter une configuration `.mjmlconfig` et les mêmes paquets que lors du rendu original. Un JSON parfaitement formé peut donc échouer à la compilation si le composant correspondant n'est pas chargé.

14. Workflow recommandé complet

Étape	Action
1	Conserver le fichier <code>.mjml</code> original comme source.
2	Convertir avec <code>mjml-parser-xml</code> .
3	Écrire le JSON avec <code>JSON.stringify(..., null, 2)</code> .
4	Valider la syntaxe JSON.
5	Modifier ou enrichir l'objet JSON.

- 6 Compiler le JSON avec mjml2html.
- 7 Examiner les erreurs MJML.
- 8 Ouvrir le HTML dans un navigateur.
- 9 Tester le HTML dans les clients de messagerie visés.
- 10 Archiver ensemble MJML, JSON, HTML et ressources.

15. Script complet : MJML → JSON → HTML

Le script suivant réalise toute la chaîne dans un seul fichier. Il crée d'abord le JSON, puis compile ce même objet vers HTML.

```
import fs from "node:fs";
import parse from "mjml-parser-xml";
import mjml2html from "mjml";

const input = process.argv[2];

if (!input) {
  console.error("Usage : node pipeline.mjs fichier.mjml");
  process.exit(1);
}

const base = input.replace(/\.mjml$/i, "");
const source = fs.readFileSync(input, "utf8");

// 1. MJML -> objet JavaScript
const tree = parse(source);

// 2. Objet -> JSON
fs.writeFileSync(
  `${base}.json`,
  JSON.stringify(tree, null, 2),
  "utf8"
);

// 3. Objet JSON MJML -> HTML
const result = await mjml2html(tree, {
  validationLevel: "soft"
});

if (result.errors?.length) {
  console.error("Erreurs MJML :", result.errors);
}

fs.writeFileSync(
  `${base}.html`,
  result.html,
  "utf8"
);

console.log(`Créé : ${base}.json`);
console.log(`Créé : ${base}.html`);
```

Installation des dépendances :

```
npm install mjml mjml-parser-xml
```

Exécution :

```
node pipeline.mjs DE_EPS_01.mjml
```

16. Diagnostic des erreurs fréquentes

Symptôme	Cause probable	Correction
JSON.parse échoue	JSON mal formé	Vérifier guillemets, virgules et accolades.
mjml2html refuse l'objet	Structure non native MJML	Vérifier tagName, attributes, children/content.
Image absente	Chemin local invalide ou fichier absent	Conserver l'arborescence ou utiliser une URL HTTPS.
Composant inconnu	Composant personnalisé non chargé	Installer/configurer le composant.
mj-include absent	Dépendance non résolue	Configurer les chemins et conserver les fichiers inclus.
Le JSON viewer ne montre pas l'email	Comportement normal	Compiler le JSON vers HTML avec mjml2html.

17. Recommandations d'archivage

- Conserver le .mjml comme source humaine principale.
- Conserver le .json comme représentation structurée destinée aux traitements automatiques.
- Conserver le .html comme sortie compilée et visualisable.
- Conserver le dossier assets avec les images et autres ressources.
- Enregistrer la version de MJML et de Node.js utilisée pour une reproduction exacte.
- Utiliser un hash SHA-256 — Secure Hash Algorithm 256-bit — si l'intégrité des fichiers doit être démontrée.

18. Résumé opérationnel

La conversion MJML vers JSON n'est pas une conversion graphique : elle transforme une arborescence de balises en arborescence d'objets. En 2026, le chemin technique privilégié pour un nouveau workflow est MJML → mjml-parser-xml → objet JavaScript → JSON.stringify → fichier JSON. Pour voir l'email, le JSON est ensuite passé directement à mjml2html, ce qui produit le HTML final. Le JSON devient particulièrement utile lorsque le contenu doit être modifié automatiquement, versionné, comparé, alimenté par une base de données ou généré par des agents logiciels.

19. Sources et références

Sources consultées le 8 octobre 2026.

- MJML Guide officiel — Documentation générale et section « Using MJML in JSON » — <https://documentation.mjml.io/>
- Dépôt officiel MJML — options, usage programmatique, validation et moteur — <https://github.com/mjmlio/mjml>
- Organisation NPM MJML — paquets actuels, dont mjml-parser-xml — <https://www.npmjs.com/org/mjml>
- Dépôt historique mjml2json — conversion MJML XML vers JSON — <https://github.com/mjmlio/mjml2json>
- Dépôt historique json2mjml — conversion JSON vers MJML XML — <https://github.com/mjmlio/json2mjml>

État des versions. Les paquets MJML évoluent. Pour une installation durable, noter les versions effectivement installées avec `npm list` et retester la compilation après toute mise à jour majeure.

FIN DU MANUEL