Équipe d'ingénierie de l'Internet (IETF)
**Request for Comments : 8259**
RFC rendue obsolète : 7159
Catégorie : Sur la voie de la normalisation
ISSN: 2070-1721

T. Bray, éditeur, Textuality
décembre 2017

Traduction Claude Brière de L'Isle

# Format d'échange de données en notation d'objet JavaScript (JSON)

**Résumé**
La notation d'objet JavaScript (JSON, *JavaScript Object Notation*) est un format d'échange de données léger, fondé sur le texte, indépendant du langage. Il est dérivé de la norme du langage de programmation ECMAScript. JSON définit un petit ensemble de règles de formatage pour la portabilité des représentations de données structurées.

Le présent document supprime des incohérences avec les autres spécifications de JSON, répare les erreurs de spécification, et propose des directives pour l'interopérabilité fondées sur l'expérience.

**Statut de ce mémoire**
Ceci est un document de l'Internet sur la voie de la normalisation.

Le présent document a été produit par l'équipe d'ingénierie de l'Internet (IETF). Il représente le consensus de la communauté de l'IETF. Il a subi une révision publique et sa publication a été approuvée par le groupe de pilotage de l'ingénierie de l'Internet (IESG). Tous les documents approuvés par l'IESG ne sont pas candidats à devenir une norme de l'Internet ; voir la Section 2 de la RFC7841.

Les informations sur le statut actuel du présent document, tout errata, et comment fournir des réactions sur lui peuvent être obtenues à http://www.rfc-editor.org/info/rfc8259.



## Table des Matières

# 1. Introduction

La notation d'objet JavaScript (JSON, *JavaScript Object Notation*) est un format de texte pour la mise en série de données structurées. Elle est dérivée des littéraux d'objets de JavaScript, comme défini dans la norme de langage de programmation ECMAScript, troisième édition [ECMA-262].

JSON peut représenter quatre types de primitives (chaînes, nombres, booléens, et nul) et deux types structurés (objets et matrices).

Une chaîne est une séquence de zéro, un ou plusieurs caractères Unicode [UNICODE]. Noter que cette citation fait référence à la dernière version de Unicode plutôt qu'à une livraison spécifique. On ne prévoit pas que de futurs changements de la spécification Unicode aient un impact sur la syntaxe de JSON.

Un objet est une collection non ordonnée de zéro, une ou plusieurs paires de nom/valeurs, où un nom est une chaîne, et où une valeur est une chaîne, un nombre, un booléen, un nul, un objet, ou une matrice.

Une matrice est une séquence ordonnée de zéro, une ou plusieurs valeurs.

Les termes "objet" et "matrice" viennent des conventions de JavaScript.

Les buts de conception de JSON étaient qu'elle soit minimale, portable, textuelle, et un sous ensemble de JavaScript.

## 1.1 Conventions et terminologie

Les mots clés "DOIT", "NE DOIT PAS", "EXIGE", "DEVRA", "NE DEVRA PAS", "DEVRAIT", "NE DEVRAIT PAS", "RECOMMANDE", "PEUT", et "FACULTATIF" en majuscules dans ce document sont à interpréter comme décrit dans le BCP 14, [RFC2119], [RFC8174] quand, et seulement quand ils apparaissent tout en majuscules, comme montré ci-dessus.

Les règles grammaticales de ce document sont à interpréter comme décrit dans la [RFC5234].

## 1.2 Spécifications de JSON

Le présent document remplace la [RFC7159]. La [RFC7159] rendait obsolète la [RFC4627], qui décrivait à l'origine JSON et enregistrait le type de support "application/json".

JSON est aussi décrit dans [ECMA-404].

La référence à [ECMA-404] dans la phrase précédente est normative, non au sens usuel que les mises en œuvre doivent la consulter afin de comprendre le présent document, mais pour souligner qu'il n'y a pas d'incohérence dans la définition du terme "texte JSON" dans ses spécifications. Noter cependant, que [ECMA-404] permet plusieurs pratiques que la présente spécification recommande d'éviter dans l'intérêt d'une interopérabilité maximale.

L'intention est que la grammaire soit la même entre les deux documents, bien que des descriptions différentes soient utilisées. Si une différence est trouvée entre eux, ECMA et l'IETF travailleront ensemble à mettre à jour les deux documents.

Si une erreur est trouvée dans l'un ou l'autre document, l'autre devrait être examiné pour voir si il y a une erreur similaire, et si cela est, elle devrait être corrigée, si possible.

Si l'un ou l'autre document est modifié à l'avenir, ECMA et l'IETF travailleront ensemble pour s'assurer que les deux restent alignés malgré le changement.

### 1.3 Introduction à cette révision

Dans les années qui ont suivi la publication de la RFC 4627, JSON a trouvé une très large utilisation. Cette expérience a révélé certains schémas qui, bien que permis par ses spécifications, ont causé des problèmes d'interopérabilité.

Aussi, un petit nombre d'errata ont été rapportés concernant la RFC 4627 (voir les identifiants d'errata [Err607] et [Err3607]) et concernant la RFC 7159 (voir les identifiants d'errata de [Err3915], [Err4264], [Err4336], et [Err4388]).

Le but du présent document est d'appliquer les errata, de supprimer les incohérences avec les autres spécifications de JSON, et de mettre en lumière les pratiques qui peuvent conduire à des problèmes d'interopérabilité.

## 2. Grammaire de JSON

Un texte JSON est une séquence de jetons. L'ensemble de jetons inclut six caractères structuraux, chaînes, numéros, et trois noms littéraux.

Un texte JSON est une valeur mise en série. Noter que certaines spécifications antérieures de JSON contraignaient un texte JSON à être un objet ou une matrice. Les mises en œuvre qui ne génèrent que des objets ou des matrices où un texte JSON est invoqué seront interopérables dans ce sens que toutes les mises en œuvre les accepteront comme textes JSON conformes.

```
JSON-text = ws value ws
```

Voici les six caractères structuraux :

```
begin-array = ws %x5B ws        ; [ crochet gauche
begin-object  = ws %x7B ws      ; { accolade gauche
end-array = ws %x5D ws          ; ] crochet droit
end-object = ws %x7D ws         ; } accolade droite
name-separator  = ws %x3A ws    ; : deux points
value-separator = ws %x2C ws    ; , virgule
```

Les espaces blanches *(ws, white space)* non significatives sont permises avant ou après les six caractères structuraux.

```
ws = *(
        %x20 /          ; espace
        %x09 /          ; tabulation horizontale
        %x0A /          ; saut à la ligne ou nouvelle ligne
        %x0D )          ; retour chariot
```

## 3. Valeurs

Une valeur JSON DOIT être un objet, matrice, nombre, ou chaîne, ou un des trois noms littéraux suivants : false *(faux)*, null *(nul)*, true *(vrai)*.

Les noms littéraux DOIVENT être en minuscules. Aucun autre nom littéral n'est permis.

```
value = false / null / true / object / array / number / string

false = %x66.61.6c.73.65   ; false

null  = %x6e.75.6c.6c      ; null

true  = %x74.72.75.65      ; true
```

## 4. Objets

Une structure d'objet est représentée comme une paire d'accolades entourant zéro, une ou plusieurs paires (ou membres) nom/valeur. Un nom est une chaîne. Un seul caractère deux points vient après chaque nom, séparant le nom de la valeur. Une seule virgule sépare une valeur du nom suivant. Les noms au sein d'un objet DEVRAIENT être uniques.

objet = début-objet [ membre *( membre séparateur-de-valeur ) ] fin d'objet

membre = chaîne séparateur-de-nom valeur

Un objet dont les noms sont tous uniques est interopérable au sens où toutes les mises en œuvre logicielles qui reçoivent cet objet vont s'accorder sur les transpositions nom-valeur. Quand les noms au sein d'un objet ne sont pas uniques, le comportement du logiciel qui reçoit un tel objet est imprévisible. De nombreuses mises en œuvre rapportent seulement la dernière paire nom/valeur. D'autres mises en œuvre rapportent une erreur ou échouent à analyser l'objet, et d'autres encore rapportent toutes les paires nom/valeur, incluant les dupliqués.

Il a été observé que les bibliothèques d'analyse JSON diffèrent quant à rendre visible l'ordre des membres objets au logiciel appelant. Les mises en œuvre dont le comportement ne dépend pas de l'ordre des membres seront interopérables au sens où elles ne seront pas affectées par ces différences.

## 5. Matrices

Une structure de matrice est représentée par des crochets entourant zéro, une ou plusieurs valeurs (ou éléments). Les éléments sont séparés par des virgules.

matrice = début de matrice [ valeur *( séparateur-de-valeur valeur ) ] fin de matrice

Il n'est pas exigé que les valeurs dans une matrice soient du même type.

## 6. Nombres

La représentation des nombres est similaire à celle utilisée dans la plupart des langages de programmation. Un nombre est représenté en base 10 en utilisant les chiffres décimaux. Elle contient un composant entier qui peut être préfixé d'un signe moins facultatif, qui peut être suivi par une partie fractionnaire et/ou une partie exposant. Les zéros en tête ne sont pas permis.

Une partie fractionnaire est un point décimal suivi par un ou plusieurs chiffres.

Une partie exposant commence par la lettre E en majuscule ou minuscule, qui peut être suivie par un signe plus ou moins. Le E et le signe facultatif sont suivis par un ou plusieurs chiffres.

Les valeurs numériques qui ne peuvent pas être représentées dans la grammaire ci-dessous (comme Infini et NaN) ne sont pas permises.

```
number = [ minus ] int [ frac ]          [ moins ] entier ] [ exposant ]
[ exp ]
decimal-point = %x2E                     ; . pour exprimer la virgule
digit1-9 = %x31-39                       ; 1 à 9
e = %x65 / %x45                          ; e à E
exp = e [ minus / plus ] 1*DIGIT         e [ moins ou plus ]  un ou plusieurs chiffres
frac = decimal-point 1*DIGIT             un ou plusieurs chiifres suivant la virgule, exprimée par un point
int = zero / ( digit1-9 *DIGIT )         entier composé de chiffres
minus = %x2D                             ; - signe moins
plus = %x2B                              ; + signe plus
zero = %x30                              ; 0 zéro
```

La présente spécification permet la mise en œuvre de limites à la gamme et la précision des nombres acceptés. Comme le logiciel qui met en œuvre les nombres IEEE 754 binary64 (à double précision) [IEEE754] est généralement disponible et

largement utilisé, une bonne interopérabilité peut être réalisée par les mises en œuvre qui n'attendent pas plus de précision ou gammes que celles fournies, en ce sens que les mises en œuvre vont approximer les nombres JSON dans la précision attendue. Un nombre JSON comme 1E400 ou 3,14159265358979323846264338327 peut présenter de potentiels problèmes d'interopérabilité, car cela suggère que le logiciel qui l'a créé s'attend à recevoir d'un logiciel qui a de plus grandes capacités pour les magnitudes et précisions numériques que ce qui est couramment disponible.

Noter que lorsque un tel logiciel est utilisé, les nombres qui sont entiers et sont dans la gamme de [-(2**53)+1, (2**53)-1] sont interopérables en ce sens que les mises en œuvre vont se mettre d'accord exactement sur leur valeur numérique.

## 7. Chaînes

La représentation des chaînes est similaire aux conventions utilisées dans la famille des langages C de programmation. Une chaîne commence et se termine par des guillemets. Tous les caractères Unicode peuvent être placés entre guillemets, sauf les caractères qui DOIVENT être échappés : les guillemets, la barre oblique inverse, et les caractères de contrôle (de U+0000 à U+001F).

Tout caractère peut être échappé. Si le caractère est dans le plan multilingue de base (de U+0000 à U+FFFF) il peut alors être représenté comme une séquence de six caractères : une barre oblique inverse, suivie par la lettre minuscule "u", suivie par quatre chiffres hexadécimaux qui codent le codet du caractère. Les lettres hexadécimales A à F peuvent être en majuscules ou en minuscules. Donc, par exemple, une chaîne contenant seulement un caractère barre oblique inverse peut être représentée par "\u005C".

Autrement, il y a des représentations d'échappement de séquence de deux caractères de certains caractères populaires. Ainsi, par exemple, une chaîne contenant seulement un caractère barre oblique inverse peut être représentée de façon plus compacte par "\\".

Pour échapper un caractère étendu qui n'est pas dans le plan multilingue de base, le caractère est représenté par une séquence de 12 caractères, codant la paire UTF-16 subrogée. Ainsi, par exemple, une chaîne contenant seulement le caractère "G clef" (U+1D11E) peut être représentée par "\uD834\uDD1E".

```
string = quotation-mark *char quotation-mark          ; chaîne = " nombre quelconque de caractères "

char = unescaped /                  ; (non échappé)
       escape (                     ; (échappé)
          %x22 /                    ; "    guillemets  U+0022
          %x5C /                    ; \    barre oblique inverse U+005C
          %x2F /                    ; /    barre oblique  U+002F
          %x62 /                    ; b    espace arrière      U+0008
          %x66 /                    ; f    saut de mot       U+000C
          %x6E /                    ; n    saut à la ligne U+000A
          %x72 /                    ; r    retour chariot U+000D
          %x74 /                    ; t    tabulation          U+0009
          %x75 4HEXDIG )            ; uXXXX             U+XXXX
escape = %x5C                       ; \
quotation-mark = %x22               ; "

unescaped = %x20-21 / %x23-5B / %x5D-10FFFF
```

## 8. Questions de chaînes et de caractères

### 8.1 Codage de caractères

Le texte JSON échangé entre des systèmes qui ne font pas partie d'un écosystème clos DOIVENT être codés en utilisant UTF-8 [RFC3629].

Les précédentes spécifications de JSON n'exigeaient pas l'utilisation de UTF-8 lors de la transmission de texte JSON. Cependant, la grande majorité des mises en œuvre de logiciel fondées sur JSON ont choisi d'utiliser le codage UTF-8, dans la mesure où c'est le seul codage qui réalise l'interopérabilité.

Les mises en œuvre NE DOIVENT PAS ajouter de marque d'ordre des octets (U+FEFF) au début d'un texte JSON transmis

sur le réseau. Dans l'intérêt de l'interopérabilité, les mises en œuvre qui analysent les textes JSON PEUVENT ignorer la présence d'une marque d'ordre des octets plutôt que de la traiter comme une erreur.

### 8.2 Caractères Unicode

Lorsque toutes les chaînes représentées dans un texte JSON sont composées entièrement de caractères Unicode [UNICODE] (éventuellement échappés) ce texte JSON est alors interopérable au sens où toutes les mises en œuvre de logiciel qui l'analysent vont s'accorder sur le contenu des noms et la valeur des chaînes dans les objets et matrices.

Cependant, l'ABNF de la présente spécification permet que les noms membres et les valeurs de chaîne contiennent des séquences binaires qui ne peuvent pas coder des caractères Unicode ; par exemple, "\uDEAD" (seul substitut UTF-16 sans correspondant). On a observé des instances de cela, par exemple, quand une bibliothèque tronque une chaîne UTF-16 sans vérifier si la coupure casse une paire de substitution. Le comportement du logiciel qui reçoit des textes JSON contenant de telles valeurs est imprévisible ; par exemple, les mises en œuvre peuvent retourner des valeurs différentes pour la longueur d'une valeur de chaîne ou même subir des exceptions fatales au moment du démarrage.

### 8.3 Comparaison de chaînes

Il est normalement exigé des mises en œuvre de logiciel qu'elles vérifient l'égalité des noms des membres objets. Les mises en œuvre qui transforment la représentation textuelle en séquences d'unités de code Unicode et effectuent ensuite une comparaison numérique, unité de code par unité de code, sont interopérables au sens où les mises en œuvre vont s'accorder dans tous les cas sur l'égalité ou l'inégalité de deux chaînes. Par exemple, les mises en œuvre qui comparent des chaînes avec des caractères échappés non convertis peuvent à tort trouver que "a\\b" et "a\u005Cb" ne sont pas égaux.

## 9. Analyseurs

Un analyseur JSON transforme un texte JSON en une autre représentation. Un analyseur JSON DOIT accepter tous les textes qui se conforment à la grammaire JSON. Un analyseur JSON PEUT accepter des formes ou extensions non JSON.

Une mise en œuvre peut fixer des limites à la taille des textes qu'elle accepte. Une mise en œuvre peut fixer des limites à la profondeur maximum d'incorporation. Une mise en œuvre peut fixer des limites à la gamme et la précision des nombres. Une mise en œuvre peut fixer des limites à la longueur et au contenu des caractères des chaînes.

## 10. Générateurs

Un générateur JSON produit des textes JSON. Le texte résultant DOIT strictement se conformer à la grammaire JSON.

## 11. Considérations relatives à l'IANA

Le type de support pour le texte JSON est application/json.

Nom du type : application
Nom de sous type : json
Paramètres requis : non applicable
Paramètres facultatifs : non applicable
Considérations de codage : binaire
Considérations de sécurité : voir la Section 12 de la RFC 8259.
Considérations d'interopérabilité : décrites dans la RFC 8259
Spécification publiée : RFC 8259

Applications qui utilisent ce type de support : JSON a été utilisé pour échanger des données entre des applications écrites dans tous les langages de programmation ActionScript, C, C#, Clojure, ColdFusion, Common Lisp, E, Erlang, Go, Java, JavaScript, Lua, Objective CAML, Perl, PHP, Python, Rebol, Ruby, Scala, et Scheme.

Informations supplémentaires :
Numéro magique : non applicable
Extension de fichier : .json

Code de type de fichier Macintosh : TEXT
Adresse personnelle & de messagerie à contacter pour plus d'informations : IESG <iesg@ietf.org>
Usage prévu : commun
Restrictions d'usage : aucune
Auteur : Douglas Crockford <douglas@crockford.com>
Contrôleur des changements : IESG <iesg@ietf.org>

Note : Aucun paramètre "charset" n'est défini pour cet enregistrement. En ajouter un n'a pas d'effet sur les receveurs conformes.

## 12. Considérations sur la sécurité

Il y a généralement des problèmes de sécurité avec les langages de script. JSON est un sous ensemble de JavaScript mais exclut les allocations et l'invocation.

Comme la syntaxe de JSON est empruntée à JavaScript, il est possible d'utiliser la fonction "eval()" de ce langage pour analyser la plupart des textes JSON (mais pas tous ; certains caractères comme U+2028 LINE SEPARATOR et U+2029 PARAGRAPH SEPARATOR sont légaux dans JSON mais pas dans JavaScript). Cela constitue généralement un risque de sécurité inacceptable, car le texte pourrait contenir du code exécutable avec les déclarations de données. La même considération s'applique à l'utilisation des fonctions de style eval() dans tout autre langage de programmation dans lequel des textes JSON se conforment à cette syntaxe de langage.

## 13. Exemples

Voici un objet JSON :

```
{
 "Image": {
    "Width":  800,
    "Height": 600,
    "Title":  "Vue du 15ème étage",
    "Thumbnail": {
       "Url":    "http://www.exemple.com/image/481989943",
       "Height": 125,
       "Width":  100
    },
    "Animated" : false,
    "IDs": [116, 943, 234, 38793]
  }
}
```

Son membre Image est un objet dont le membre Thumbnail est un objet et dont les identifiants membres sont une matrice de nombres.

Voici une matrice JSON qui contient deux objets :

```
[
  {
    "precision": "zip",
    "Latitude":  37.7668,
    "Longitude": -122.3959,
    "Address":   "",
    "City":      "SAN FRANCISCO",
    "State":     "CA",
    "Zip":       "94107",
    "Country":   "US"
  },
  {
    "precision": "zip",
```

```
      "Latitude":  37.371991,
      "Longitude": -122.026020,
      "Address":   "",
      "City":      "SUNNYVALE",
      "State":     "CA",
      "Zip":       "94085",
      "Country":   "US"
    }
  ]
```

Voici trois petits textes JSON qui contiennent seulement des valeurs:

"Hello world!"

42

true

## 14. Références

### 14.1 Références normatives

[ECMA-404] Norme ECMA-404, "The JSON Data Interchange Format", < http://www.ecma-international.org/publications/standards/Ecma-404.htm >.

[IEEE754] IEEE, "IEEE Standard for Floating-Point Arithmetic", IEEE 754.

[RFC2119] S. Bradner, "Mots clés à utiliser dans les RFC pour indiquer les niveaux d'exigence", BCP 14, mars 1997. *(MàJ par* RFC8174) DOI 10.17487/RFC2119.

[RFC3629] F. Yergeau, "UTF-8, un format de transformation de la norme ISO 10646", STD 63, novembre 2003, DOI 10.17487/RFC3629.

[RFC5234] D. Crocker, éd., P. Overell, "BNF augmenté pour les spécifications de syntaxe : ABNF", janvier 2008. (STD0068), DOI 10.17487/RFC5234.

[RFC8174] B. Leiba, "Ambiguïté des mots clés en majuscules ou minucules dans la RFC2119", mai 2017. BCP14. *(MàJ* RFC2119), DOI 10.17487/RFC8174.

[UNICODE] Unicode Consortium, "The Unicode Standard", < http://www.unicode.org/versions/latest/ >.

### 14.2 Références pour information

[ECMA-262] Norme ECMA-262, "ECMAScript Language Specification", troisième édition, décembre 1999, <http://www.ecma-international.org/publications/files/ECMA-ST-ARCH/ECMA-262,%203rd%20edition,%20December%201999.pdf >.

[Err607] Erratum ID 607, RFC 4627, < https://www.rfc-editor.org/errata/eid607 >.

[Err3607] Erratum ID 3607, RFC 4627, < https://www.rfc-editor.org/errata/eid3607 >.

[Err3915] Erratum ID 3915, RFC 7159, < https://www.rfc-editor.org/errata/eid3915 >.

[Err4264] Erratum ID 4264, RFC 7159, < https://www.rfc-editor.org/errata/eid4264 >.

[Err4336] Erratum ID 4336, RFC 7159, < https://www.rfc-editor.org/errata/eid4336 >.

[Err4388] Erratum ID 4388, RFC 7159, < https://www.rfc-editor.org/errata/eid4388 >.

[RFC4627] D. Crockford, "Type de support application/json pour la notation d'objet JavaScript (JSON)", juillet 2006. *(Information ; Remplacée par* RFC7159), DOI 10.17487/RFC4627.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <https://www.rfc-editor.org/info/rfc7159>.

## Appendice A. Changements à la RFC 7159

Cette section fait la liste des changements apportés au texte de la RFC 7159.

- Le paragraphe 1.2 a été mis à jour pour refléter la suppression d'une spécification JSON de ECMA-262, pour faire de ECMA-404 une référence normative, et pour expliquer la signification particulière de "normative".
- Le paragraphe 1.3 a été mis à jour pour refléter les errata de la RFC 7159, mais pas de la RFC 4627.
- Le paragraphe 8.1 a été changé pour demander l'utilisation de UTF-8 à la transmission sur le réseau.
- La Section 12 a été mise à jour pour préciser la description du risque pour la sécurité qui découle de l'utilisation de la fonction ECMAScript "eval()".
- Le paragraphe 14.1 a été mis à jour pour inclure ECMA-404 comme référence normative.
- Le paragraphe 14.2 a été mis à jour pour retirer ECMA-404, metre à jour la version de ECMA-262, et rafraîchir la liste des errata.

**Contributeurs**

La RFC 4627 a été écrite par Douglas Crockford. Le présent document a été construit en faisant un nombre relativement faible de modifications à ce document, et donc, la grande majorité du texte présenté ici est le sien.

**Adresse de l'auteur**

Tim Bray (editor)
Textuality
mél : tbray@textuality.com