JSON (*Javascript Object Notation*)

JSON (2002)

- Un format **textuel** et léger de représentation de données **structurée**
  - **Lisible** par l'homme
  - Généré et analysé **trivialement** par la machine
- **Inspiré** largement par le **javascript** (sous-ensemble de ECMA-262) tout en restant indépendant du langage : un sous-ensemble minimal & portable de javascript

JSON (RFC 4627 en 2006)

- Beaucoup de langages supportés : objective C, C, C++, java ect...
- Beaucoup de langues : français, anglais, chinois, japonnais...
- Il n'existe qu'une seule version de JSON : aucun révision prévue

Usage

- JSON est utilisé
  - En tant que format d'échange entre un client (typiquement un navigateur, possiblement embarqué) et un serveur (backend),
  - Non pas en tant que format de représentation de données
  - Par google, yahoo (services Web)
  - Dans le domaines des smartphones du fait de sa légéreté...

JSON – Exemple

```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]}}}
```

Format – 2 types structurés

- JSON s'organise autours de **deux structures** (ou types structurés)
  - `Object` : une collection (non ordonnée) de paires (nom-valeur)
  - `Array` : un séquence (ordonnée) de valeurs
- Utilisées par n'importe quel langage moderne, ces 2 types permettent de représenter des structures, des tables (par exemple hash tables) ect...
- L'imbrication est permise mais aucune structure récursive n'est supportée

Format – `array`

- Une séquence ordonnée de 0 à n valeurs encerclée par [ ]
  - `array =  [ value *( , value ) ]`
- Est utilisée lorsque les noms (qui sont uniques) sont des entiers séquentiels
- Exemple : [30 ,40, 50]

Format - `object`

- Une collection non ordonnées de 0 à n paires
  - `object = [ member *( , pair ) ]`
  - Chaque paire est composée d'un nom et d'une valeur
    - `pair = string : value`
    - `nomDeLaPaire1 : valeurDeLaPaire1 , nomDelaPaire2 : valeurDeLaPaire2`
- Est souvent utilisé lorsque les noms sont des strings
- Exemple : `{"size": 50}`

Format - Exemple d'objects imbriqués

```
{"Image": {
    "Width":  800,
    "Height": 600,
    "Title":  "View from 15th",
    "Thumbnail": {
        "Url":"http://ex.fr/img/48",
        "Height": 125,
        "Width":  "100"},
    "IDs": [116, 943, 234, 38793]}}
```

Format - Exemple de tableau contenant 2 objets

```
[   {  "Latitude":  37.7668,
       "Longitude": -122.3959,
       "Address":   "",
       "City": "SAN FRANCISCO",
       "Country":   "US" },
     {  "Latitude":  37.371991,
       "Longitude": -122.026020,
       "City":      "SUNNYVALE",
       "Country":   "US"}   ]
```

Format – types primitifs

- Aux 2 types structurés (`array, object`) s'ajoutent 4 types primitifs

string	Séquence de 0 à n caractères unicodes suivant les conventions du langage de programmation C, entouré par “”
number	Nombre décimale signé
boolean	true ou false
null	null

- Une valeur est de type structuré ou primitive

Format – type primitif string

- Suite de 0 à n caractères unicodes
- Utilise la représentation utilisée dans le langage C
- Les caractère unicodes sont entourés de “” à l'exception des caractères d'échappement : ? , \, et caractères de contrôle (U+0000 à U+001F)

Format – type string

- Le type string encodé au format unicode utilise une représentation identique au langage C

```
string = quotation-mark *char quotation-mark

 char = unescaped / escape (

 %x22 /          ; "    quotation mark  U+0022

%x5C /          ; \    reverse solidus U+005C

%x2F /          ; /    solidus         U+002F

%x62 /          ; b    backspace       U+0008

%x66 /          ; f    form feed       U+000C
```

Format – type string

```
%x6E /          ; n    line feed       U+000A
%x72 /          ; r    carriage return U+000D
%x74 /          ; t    tab             U+0009
%x75 4HEXDIG )  ; uXXXX                U+XXXX
escape = %x5C              ; \
quotation-mark = %x22      ; "
unescaped = %x20-21 / %x23-5B / %x5D-
10FFFF
```

Format – type primitif number

- Entier, possiblement signé, contenant ou non une fraction (partie décimale séparée par un point suivi d'au moins un nombre) ou/et un exposant (lettre e en majuscule ou minuscule suivie de l'exposant possiblement négatif)
- Formes octale ou décimale non permises

```
number = [minus] int [frac] [exp]
         exp = e/E [ - / + ] 1*DIGIT
         frac = . 1*DIGIT
         int = 0 / ( 1-9 *DIGIT )
```

Générateur & Parseur

- Le **générateur** doit **créer un texte conforme** à la grammaire JSON
- Le **parseur** transforme un texte en une autre représentation. Il annalyse tout texte conforme à la grammaire JSON et possiblement des **extensions**, des formats non conformes
- Les limites (longueur, profondeur du texte, des nombres, des strings et de leur contenus) sont fixées au niveau de l'implémentation

Conclusion

- JSON est utilisé en tant que format léger d'échange entre un client et un serveur
- JSON constitue une alternative à XML notamment lorsque les clients (smartphones) sont contraints
  - Est géré par les navigateurs actuels
  - Est intégré par les plateformes des services Web
- De multiples implémentations / languages (Unicode) sont disponibles

Biblio&Webgraphie

www.json.org	Présentation	Exemples + librairies
http://www.ietf.org/rfc/rfc4627	RFC 4627 de l'IETF	
http://www.unicode.org/charts/	Ensemble des documentations sur le standard unicode	
www.eclipse.org http://www.eclipse.org/webtools/	Eclipse Web service tools	

Rappel – Unicode

- Il existe de nombreux standards industriel (par ex. L'encodage pdf), nationaux ou régionaux tels que ISO-latin-1/9 pour l'Europe de l'ouest
- Unicode est le standard d'encodage multilingue des caractères

Rappel – encodage Unicode

- Le caractère ASCII comme point de départ
- Ne nécessite pas d'utiliser des caractères de contrôle pour spécifier un caractère dans un langage “exotique”
- 3 formats d'encodage sont utilisés 8, 16, 32 bits (8 bits en accord avec le système ASCII)
- Un sur-ensemble des formats d'encodage ISO existants répertoriant les caractères, définissant leur encodage, leur sérialisation

Rappel – encodage Unicode

- Un caractère abstrait est :
  - définis en tant que plus petit composant du langage ayant une valeur sémantique
  - Nommé et répertorié
    - les caractères d'un même scripte sont groupés
    - Un caractère peut avoir plusieurs codes
  - Encodé sous la forme d'un numéro allant de 0 à 0010FFFF préfixé par un “+U”pour des raisons de compatibilité avec UTF-16, soit en binaire un valeur maximale de `0000 0000 0001 0000 1111 1111 1111 1111`)
  - Sérialisé (par octet)

Rappel – encodage Unicode

- Un caractère faisant partie du plan multilingue basique (de U+0000 à U+FFFF) est représenté par une séquence de 6 caractères
- \uxxxx avec xxx correspondant au code héxadécimale du caractère
- Exemple : "\u005C"
- L'ordre des caractères suit celui des alphabets
- Un caractère est défini par des propriétés (directives visuelles)

Rappel – encodage Unicode

- Il existe 3 formats d'encodage
  - UTF-32 (appelé aussi USC4) : numéro sur 32 bits
  - UTF-16 : numéro sur 16 bits
    - Format utilisé à l'origine
    - Un codage par indirection est utilisé pour les valeurs supérieures à 00010000
  - UTF-8 : chaque numéro est codé par une suite de 0 à 4 octets

Rappel - UTF8

- Chaque caractère est representé sous la forme d'un code composé d'au plus 4 *groupes d'un octet*

	Groupe 1	*Groupe 2*	*Groupe 3*	*Groupe 4*
000-007F	0xxx**xxxx**			
0080-07FF	110x**xxxx**	10xxxxxx		
0800-D7FF	1110**xxxx**	10xxxxxx	10xxxxxx	
E000-FFFF	1110**xxxx**	10xxxxxx	10xxxxxx	
10000-10FFFF	11110**xxx**	10xx**xxxx**	10xxxxxx	10xxxxxx

- *Le code “U+F03F” soit en binaire 1111 0000 0011 1111 est représenté en unicode par 1110 1111 1000 0000 1000 0011 soit 0xEF 0x80 0xBF*

Rappel – UTF 16

- Chaque caractère est representé sous la forme d'un code composé de 1 ou 2 *groupes de 2 octets*

	Group 1	Group 2
0000-D7FF	xxxxxxxxxxxxxxxx	
E000-FFFF	xxxxxxxxxxxxxxxx	
10000-10FFFF	110110xxxxxxxxxx	110111xxxxxxxxxx

- *Très populaire*
- *Trois encodages sont proposés : little endian, big endian,endianless*

UTF-32

- Chaque caractère est representé sous la forme d'un groupe de 4 *octets*

	Groupe 1
0000-D7FF	
E000-10FFFF	00000000000xxxxxxxxxxxxxxxxx

- Encodage coûteux
- *Trois encodages sont proposés : little endian, big endian,endianless*

Rappel – encodage Unicode

- L'ensemble des caractères unicode est organisé en plusieurs plans :
  - Plan 0 (U+0000 – U+FFFF) Basic Multilingual
  - Plan 1 (U+10000 – U+1FFFF) Supplementary Multilingual
  - Plan 2 (U+20000 - U+2FFFF) Supplementary Ideographic
  - Plane 14 (U+E0000 – U+EFFFF) Supplementary, Special-purpose
  - Plans 15 et 16 (U+F0000 - U+10FFFF) Private use

Rappel – encodage Unicode

- Il existe plusieurs types de caractères
  - Graphic (lettre, nombre, symbole, ponctuation, espace)
  - Format : caractère invisible affectant les caractères voisins (ex : séparateur de ligne)
  - Contrôle : usage défini par un protocole ou standard autre qu'Unicode
  - Private-use : usage privé (autre qu'unicode)
  - Surrogate : réservé à UTF-16
  - Non character : usage interne
  - Reserved : réservé (assignement futur) :

Rappel – encodage Unicode

- Un même caractère peut être représenté en tant que :
  - Caractère précomposé. Par exemple ü est codé U+0075
  - Caractère composite : U+0075 "u" suivi de U+0308 "¨". La composition est utile pour appliquer des traitements particuliers
- Les caractères sont canoniques lorsqu'ils ne diffèrent pas malgré leur encodage différent
- Les duplications sont évitées via une consolidation inter-langage

Rappel – encodage Unicode

- Un **même** caractère peut être affiché de plusieurs façons : **A** A A *A*
- La même lettre peut apparaître dans plusieurs alphabets et donc posséder plusieurs codes
- Seule la lettre au format brut est représentée
- Le rendu des caractère est une tâche complexe : un même caractère peut prendre plusieurs formats suivant son contexte
  - Un ordre logique de composition lié à la prononciation ou à la façon de “taper” sur le clavier

Rappel – encodage unicode

- Chaque caractère dispose
  - de propriétés facilitant la gestion du caractère
  - d'algorithmes permettant de gérer un caractère
- Propriétés et algorithmes sont stockés séparement
- Ces algorithmes permettent l'affichage (de gauche à droite par exemple) en gérant l'ordre logique de composition