
{JSON}
JavaScript Object Notation

JSON

1. Extension du fichier est **.json**
2. Il permet de stocker et d'échanger des données textuelles en utilisant la syntaxe objet de JavaScript
3. Format compréhensible par tous (humain et machine).
4. Ne dépend d'aucun langage. Il est pris en charge par de nombreux langages : JavaScript, PHP, Perl, Python, Ruby, Java,...
5. Pas de problématique de version, JSON est stable et inchangé

JSON peut prendre les formes suivantes :

1. Un objet, qui est un ensemble de couples nom/valeur non ordonnés et qui commence par { et se termine par }. Chaque nom est suivi de : et les couples nom/valeur sont séparés par , Ex:

   {"firstName":"John", "lastName":"Doe"}

2. Un tableau est une collection de valeurs ordonnées. Un tableau commence par [et se termine par]. Les valeurs sont séparées par ,

   [{ "books": [{ "language":"Java" , "edition":"second" }, { "language":"C++" , "edition":5 }, {"language":"C","edition ":3}] }]

- Une valeur peut être soit une chaîne de caractères entre guillemets, un nombre, une booléenne, un objet, un tableau ou **null**.

Imbrications: Il est possible de déclarer des tableaux d'objets, des objets contenant des tableaux ou d'autres objets, ...

```
var commandes = [
    { "client": "Jean",
      "articles": [
           { "nom": "Livre", "quantite": 2, "prix_unitaire": 21.99 } ,
           { "nom": "Stylo", "quantite": 4, "prix_unitaire": 0.79 }
      ],
      "mode_paiement": "chèque"
    },
    { "client": "Pierre",
      "articles": [
           { "nom": "Livre", "quantite": 1, "prix_unitaire": 21.99 } ,
           { "nom": "Trombones", "quantite": 50, "prix_unitaire": 0.05 }
      ],
      "mode_paiement": "espèces"
    }
];
```

JSON & JavaScript

- On peut utiliser la fonction JavaScript `JSON.parse(text)` pour convertir un text JSON à un objet JavaScript:

```
<p id="demo"></p>
<script>
var text ='{"employees":[{"firstName":"John","lastName":"Doe" },
{"firstName":"Anna","lastName":"Smith" },
{"firstName":"Peter","lastName":"Jones" }]}';
obj = JSON.parse(text);
document.getElementById("demo").innerHTML =
obj.employees[2].firstName + " " + obj.employees[2].lastName;
</script>
```

- La réciproque de la fonction `JSON.parse()` est `JSON.stringify()`, elle convertit un objet JavaScript en chaîne JSON

```
var objet = {};
objet.couleur = 'rouge';
objet.forme = 'carré';
objet.contient = ['téléphone', 'clés de voiture'];

console.log( JSON.stringify(objet) );
// {"couleur":"rouge","forme":"carré","contient":["téléphone","clés de voiture"]}
```

JSON & PHP

Pour créer un JSON facilement en PHP nous pouvons utiliser la fonction PHP `json_encode` qui convertit un array en texte formaté JSON

```php
$data = array();
$data["name"]  = "samir";
$data["date"]  = date("Y.m.d");
$data["admin"] = true;
echo json_encode( $data ).'</br>';
// Affichera:
// {"name":"samir","date":"2017.11.21","admin":true}
```

JSON & PHP

- Pour lire des données encodées dans le format JSON vous avez à disposition la fonction `json_decode()` qui va vous permettre de convertir les données du JSON soit en objet ou soit en tableau PHP.

```php
$json_source = '{"nom":"Amine", "naissance":"1981-06-12"}';

// Décode le JSON
$json_data = json_decode($json_source);

// Affiche la valeur des attributs du JSON
echo $json_data->nom.' '.$json_data->naissance;
```

- Pour converti l'objet retourné en un tableau associatif, vous devez ajouter un deuxième argument à la fonction `json_decode()` avec la valeur **true**.

```php
$json_source = '{"nom":"Adriana", "naissance":"1981-06-12"}';
$json_data = json_decode($json_source, true);
echo $json_data['nom'].' '.$json_data['naissance'];
```

- Si le fichier JSON contient une collection de données, il est possible d'accéder un par un à chaque élément du JSON. Pour se faire, vous pouvez utiliser `foreach()` sur la variable contenant les valeurs retournées par

```php
// Les crochets qui délimitent le JSON indiquent qu'il s'agit d'une collection de données
$json_source = '[{"nom":"Adriana"}, {"nom":"Candice"}]';

// On va boucler sur un tableau
$json_data = json_decode($json_source, true);
foreach($json_data as $v){
   echo $v['nom'].'<br>';
}

// On va boucler sur un objet
$json_data = json_decode($json_source);
foreach($json_data as $v){
   echo $v->nom.'<br>';
}
```

PDO (PHP Data Objects)

- **PDO** est une extension définissant l'interface pour accéder à plusieurs types de base de données.
- Son principal avantage est qu'il permet une abstraction pour l'accès aux données. C'est-à-dire que les fonctions pour exécuter des requêtes et pour récupérer des données sont les mêmes, quelque-soit le serveur SQL utilisé (MySQ
- **PDO** e nts types de bases


```
print_r(PDO::getAvailableDrivers());
```

- L'accès à la base de données se fait en instanciant un objet **PDO**

```php
<?php
try {
  # MS SQL Server and Sybase with PDO_DBLIB
  $DBH = new PDO("mssql:host=$host;dbname=$dbname, $user, $pass");
  $DBH = new PDO("sybase:host=$host;dbname=$dbname, $user, $pass");
  # MySQL with PDO_MYSQL
  $DBH = new PDO("mysql:host=$host;dbname=$dbname", $user, $pass);
  # SQLite Database
  $DBH = new PDO("sqlite:my/database/path/database.db");
}
catch(PDOException $e) {
    echo $e->getMessage();
}
?>
```

- Selon les **drivers** , les informations pourront être légèrement différentes. Par exemple ici avec **sqlite**, seul le driver et le nom du fichier de base sont notés

Passer des requêtes en PDO

- PDO fait la distinction entre deux formes de requêtes:

  1. La méthode **exec()** permet de passer et exécuter une requête SQL de type **INSERT**, **UPDATE**, **DELETE**. Elle retourne le nombre de lignes affectées par la requête.

  2. La méthode **query()** permet de passer et exécuter une requête SQL de type SELECT. Elle retourne objet **PDOStatement**.

- Une des possibilités de **PDO**, qui est extrêmement populaire, est l'utilisation des requêtes préparées.

  1. La méthode **prepare()** permet de préparer une requête que l'on exécutera ensuite avec **PDOStatement::execute()**

Quelques méthodes de **PDOStatement**

- **PDOStatement::fetch():** récupère la ligne suivante d'un jeu de résultats **PDO**.
- **PDOStatement::fetchAll():** retourne un tableau contenant toutes les lignes du jeu d'enregistrements **PDO**.
- **PDOStatement::fetchObject():** récupère la ligne suivante et la retourne en tant qu'objet.
- **PDOStatement::rowCount():** retourne le nombre de lignes affectées par le dernier appel à la fonction.
- **PDOStatement::setFetchMode():** définit le mode de récupération par défaut pour cette requête(FETCH_ASSOC, FETCH_OBJ, etc.)

- La méthode **exec():**

```php
$db = new PDO('mysql:host=localhost;dbname=testdb', 'username', 'password');
$affected_rows = $db->exec("UPDATE table SET field='value'");
echo $affected_rows.' were affected'
```

- La méthode **query():**

```php
$resultats=$db->query("SELECT field1 FROM table ORDER BY field ASC");
$resultats->setFetchMode(PDO::FETCH_OBJ); // on dit qu'on veut que le résultat
// soit récupérable sous forme d'objet
while( $ligne = $resultats->fetch() )
{
  echo $ligne->field1.'<br />';
}
```

- La méthode **prepare**

- ✓ Cette méthode prépare une requête SQL à être exécutée en offrant la possibilité de mettre des paramètres (marqueurs) qui seront substitués lors de l'exécution.
- ✓ Il existe deux types de marqueurs qui sont respectivement anonyme (**?)** et les marqueurs nominatifs

```php
# sans marqueurs - ripe for SQL Injection!
$STH = $DBH->prepare("INSERT INTO client (name, addr, city) values ($name, $addr, $city)");

# marqueurs anonymes
$STH = $DBH->prepare("INSERT INTO client (name, addr, city) values (?, ?, ?)");

# marqueurs nominatifs
$STH = $DBH->prepare("INSERT INTO client (name, addr, city) value (:name, :addr, :city)");
```

1. Marqueurs anonyme **(Unnamed Placeholders)**:

```php
$STH->bindParam(1, $name); $STH->bindParam(2, $addr);
$STH->bindParam(3, $city);
# insert one row
$name = "Daniel"; $addr = "1 Wicked Way";
$city = "Arlington Heights";
$STH->execute();
# insert another row with different values
$name = "Steve"; $addr = "5 Circle Drive";
$city = "Schaumburg";
$STH->execute();
```

- On peut utiliser **bindValue** en spécifiant le type de chaque paramètre:

```php
$DBH = new PDO('mysql:host=localhost;dbname=testdb', 'username', 'password');
$STH = $DBH->prepare("SELECT * FROM table WHERE id=? AND name=?");
$STH->bindValue(1, $id, PDO::PARAM_INT);
$STH->bindValue(2, $name, PDO::PARAM_STR);
$STH->execute();
$rows = $STH->fetchAll(PDO::FETCH_ASSOC);
```

-

```php
# the data we want to insert
$data = array('Cathy', '9 Dark and Twisty Road', 'Cardiff');
$STH = $DBH->prepare("INSERT INTO client (name, addr, city) values (?, ?, ?)");
$STH->execute($data);
```

2. les marqueurs nominatifs (**Named Placeholders**):

```
$STH = $DBH->prepare("INSERT INTO client (name, addr, city) value (:name, :addr, :city)");
# placeholders always start with a colon.
$STH->bindParam(':name', $name); $name="Adam";
$STH->bindParam(':addr', $addr); $addr="addr1"
$STH->bindParam(':city', $city); $city="Setif";
$STH->execute();
```

- Ici, les données sont stockées dans un tableau associative:

```
# the data we want to insert
$data = array( 'name' => 'Cathy', 'addr' => '9 Dark and Twisty', 'city' => 'Cardiff' );
# the shortcut!
$STH = $DBH->("INSERT INTO Client (name, addr, city) value (:name, :addr, :city)");
$STH->execute($data);
```

- Exécuter des instructions préparées dans une boucle

```php
$values = array('bob', 'alice', 'lisa', 'john');
$name = '';
$STH = $DB->prepare("INSERT INTO table('name') VALUES(:name)");
$STH->bindValue(':name', $name, PDO::PARAM_STR);
foreach($values as $name) {
    $STH->execute();
}
```

- Une autre fonctionnalité intéressante des marqueurs nominatifs est la possibilité d'insérer des objets directement dans une base de données

```
    # a simple object
    class client {
      public $name;
      public $addr;
      public $city;
      function __construct($n,$a,$c) {
      $this->name = $n;
      $this->addr = $a;
      $this->city = $c;
      }
  # etc ...
  }
    $cathy = new person('Cathy','9 Dark and Twisty','Cardiff');
    # here's the fun part:
    $STH = $DBH->("INSERT INTO Client (name, addr, city) value (:name, :addr, :city)");
    $STH->execute((array)$cathy);
```

Exemple: PHP & SQLite

```php
<?php
try {
   # SQLite Database
  //Create (connect to) SQLite database in file
  $file_db = new PDO('sqlite:MyBase');
    // Set errormode to exceptions
    $file_db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    // Create table Client
    $file_db->exec("CREATE TABLE IF NOT EXISTS Client (
                    id INTEGER PRIMARY KEY,
                    name TEXT,
                    addr TEXT,
                    age INTEGER)");
    // Array with some test data to insert to database
    $Clients = array(
                  array('name' => 'Amine', 'addr' => 'Setif',
                        'age' => 25),
                  array('name' => 'Samir', 'addr' => 'Alger',
                        'age' => 20),
                  array('name' => 'sara',  'addr' => 'Jijel',
                        'age' => 26)
                );
  // Prepare INSERT statement
   $insert = "INSERT INTO Client (name, addr, age)
                VALUES (:name, :addr, :age)";
    $stmt = $file_db->prepare($insert);
```

```php
    // Bind parameters to statement variables
    $stmt->bindParam(':name', $name);
    $stmt->bindParam(':addr', $addr);
    $stmt->bindParam(':age', $age);

    // Loop thru all clients and execute prepared insert statement
    foreach ($Clients as $m) {
      // Set values to bound variables
      $name = $m['name'];
      $addr = $m['addr'];
      $age = $m['age'];

      // Execute statement
      $stmt->execute();
    }

      $result = $file_db->query('SELECT * FROM Client');
      while ($ligne = $result ->fetch(PDO::FETCH_ASSOC)) {
         echo 'ID: '.$ligne['id'].'<br>'.' Name: '.$ligne['name'].'<br/>'
         .' Addr: '.$ligne['addr'].'<br>'.' Age: '.$ligne['age'].'<hr>';;
       }
}
catch(PDOException $e) {
    echo $e->getMessage();
}
?>
```