Mise en œuvre du stockage distant au travers d'exemples

Cinq exemples complets de mise en œuvre vont être présentés.

Pour les besoins de ces exemples, nous réutiliserons la table MySQL `voitures` vue dans le chapitre précédent.

Rappelons simplement ici la structure de cette table :

- `code_voiture` sur quatre caractères (code de la voiture).
- `libelle_voiture` sur 20 caractères (libellé de la voiture).
- `vitesse_maximale_voiture` en entier (vitesse maximale que la voiture peut atteindre, sur circuit bien sûr !).

1. Exemple 1 : Présentation du système de notation JSON

Commentaire du script JSON_01.htm

Dans ce premier exemple, nous nous contenterons d'étudier le système de notation propre à JSON et à étudier le mode d'accès aux propriétés des objets.

L'intégralité du code JavaScript commenté ci-après est placée dans la section HTML `<body>` du script (nommé `JSON_01.htm`). Ce code est donc encadré par :

```
<!-- Début script JavaScript -->
<script type="text/JavaScript">
    /* Code JavaScript commenté ci-après */
</script>
```

Le script débute par la définition de trois objets (des voitures) caractérisés par les propriétés habituelles (cf. chapitre précédent) code, libellé et vitesse maximale :

```
/* Définition de 3 objets JSON */
var voiture1 = {
    code : "V001",
    libelle : "Porsche 930 Turbo",
    vitesse_maxi : 290
};
var voiture2 = {
    code : "V002",
    libelle : "Porsche 964 Turbo",
    vitesse_maxi : 300
};
var voiture3 = {
    code : "V003",
    libelle : "Ferrari 430",
    vitesse_maxi : 320
};
```

Les valeurs des propriétés textuelles (libelle) sont notées entre guillemets.

Créons aussi un objet composite (assemblage d'objets) de nom `allemandes` qui regroupe les objets `voiture1` et `voiture2`, qui sont effectivement des voitures allemandes, comme suit :

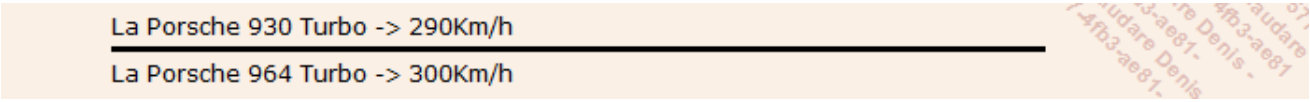
```
/* Définition d'un objet composite */
var allemandes = {
    allemande1 : voiture1,
    allemande2 : voiture2
};
```

Voyons maintenant comment utiliser ces objets en affichant par exemple leurs propriétés :

```
/* Affichage de propriétés des objets
voiture1 et voiture2 */
document.write("La " + voiture1.libelle
+ " -> " + voiture1.vitesse_maxi + "Km/h");
document.write("<hr />");
document.write("La " + voiture2.libelle
+ " -> " + voiture2.vitesse_maxi + "Km/h");
```

La notation est on ne peut plus simple, il suffit d'indiquer le nom de l'objet en préfixe, puis de signaler (après un point) la propriété souhaitée en suffixe.

L'affichage à l'écran donne ceci :

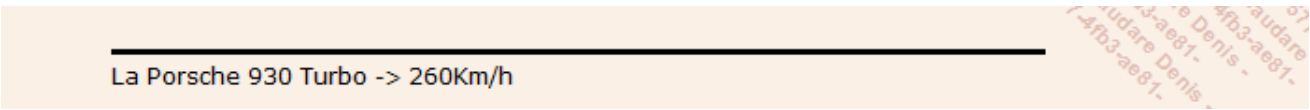


Modifions maintenant l'une des valeurs des propriétés, la vitesse maximale de la `voiture1` (290 initialement) pour la ramener à une valeur plus plausible (260) et affichons ensuite un compte rendu :

```
/* Modification de la propriété vitesse_maxi
de l'objet voiture1 */
voiture1.vitesse_maxi = 260;

/* Affichage de propriétés de l'objet voiture1 */
document.write("<hr />");
document.write("La " + voiture1.libelle
+ " -> " + voiture1.vitesse_maxi + "Km/h");
```

L'affichage est le suivant :



Nous avons déclaré un objet composite `allemandes`, voyons comment l'utiliser :

```
/* Affichages de la propriété vitesse_maxi de l'objet
voiture2 inclus dans l'objet allemandes */
document.write("<hr />");
document.write("Affichages via l'objet allemandes :");
document.write("<br />");
document.write("La " + allemandes.allemande2.libelle + " -> " +
allemandes.allemande2.vitesse_maxi + "Km/h (syntaxe 1)");
document.write("<br />");
document.write("La " + allemandes["allemande2"]["libelle"] + " -> " +
allemandes["allemande2"]["vitesse_maxi"] + "Km/h (syntaxe 2)");
```

Dans notre séquence de code, deux syntaxes d'affichage de la propriété `vitesse_maxi` sont proposées pour la `voiture2` :

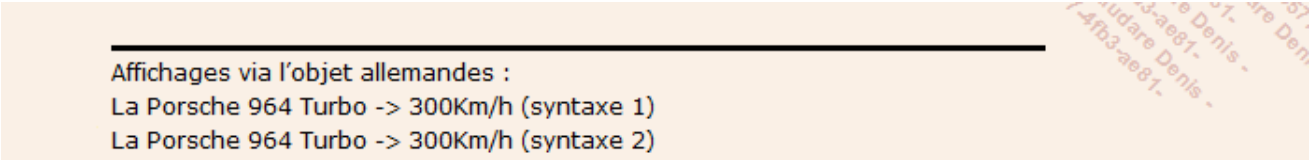
La syntaxe "objet" (pour le libellé) :

```
allemandes.allemande2.libelle
```

La syntaxe "tableau associatif" (pour le libellé) :

```
allemandes["allemande2"]["libelle"]
```

Nous obtenons à l'écran le résultat suivant :



Terminons notre exemple en étudiant la possibilité de construire des objets JSON à partir de tableaux :

```
/* Définition de 2 tableaux (libelles et vitesses_maxi) */
var libelles = [ "Porsche 930 Turbo", "Porsche 964 Turbo", "Ferrari 430" ];
var vitesses_maxi = [ 290, 300, 320 ];

/* Définition d'un objet JSON basé sur les tableaux
libelles et vitesses_maxi */
var voitures = {
      lib:libelles,
      vit:vitesses_maxi
};
```

Dans la séquence de code, un premier tableau `libelles` est créé et les trois libellés vus précédemment y sont placés. Un second tableau `vitesses_maxi` regroupe les trois valeurs de vitesse. Bien évidemment les informations sont stockées dans un ordre pertinent dans ces deux tableaux.

Enfin un objet JSON nommé `voitures` est prévu, les propriétés `lib` et `vit` sont des propriétés composites basées sur les deux tableaux `libelles` et `vitesses_maxi`.

Vous avez hâte de voir comment cela s'utilise, allons-y :

```
/* Affichage (via des tableaux) des propriétés de la 2ème voiture */
document.write("<hr />");
document.write("Affichage (via des tableaux) des propriétés de la 2ème
voiture :");
document.write("<br />");
document.write(voitures.lib[1] +" -> " + voitures.vit[1] + " Km/h");
```

La notation est aussi très intuitive, il suffit d'indiquer le nom de l'objet JSON en préfixe puis après un point de préciser le nom du tableau concerné avec le rang de la valeur souhaitée entre `[]`.

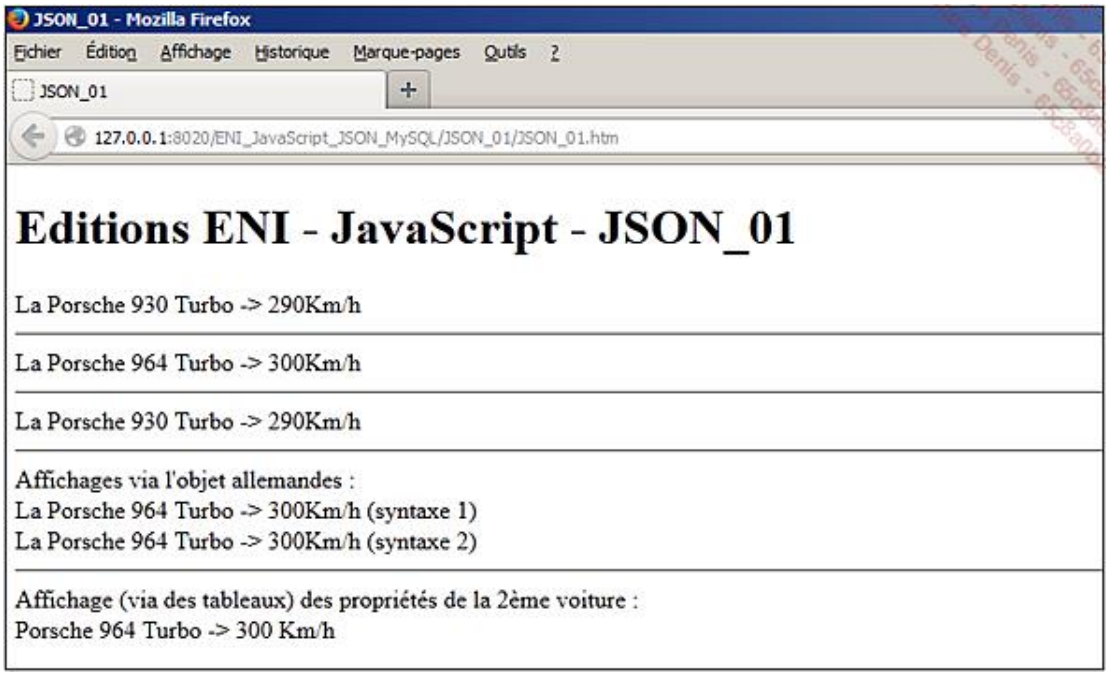
Attention dans les tableaux, le premier élément a pour valeur de rang 0.

Dans notre cas, nous obtenons l'affichage suivant :

Affichage (via des tableaux) des propriétés de la deuxième voiture :
Porsche 964 Turbo -> 300 Km/h

Compte rendu d'exécution

Des comptes rendus partiels ont été proposés dans le commentaire du script `JSON_01.htm`. Terminons par le compte rendu de l'exécution tel qu'il se présente dans notre navigateur habituel (Mozilla Firefox) :



2. Exemple 2 : Lecture d'un fichier JSON via XMLHttpRequest

Commentaire du script JSON_02.htm

Dans ce deuxième exemple, nous allons voir comment accéder au contenu d'un fichier de données JSON depuis un script JavaScript et ceci au travers d'un objet `XMLHttpRequest`.

L'objet `XMLHttpRequest` a été utilisé dans le cadre du chapitre précédent.

Pour faciliter la compréhension de l'application, reprenons notre exemple basé sur nos trois voitures de sport :

```
{
    "voiture1":{
        "code":"V001",
        "libelle":"Porsche 930 Turbo",
        "vitesse_maxi":290
    },
    "voiture2":{
        "code":"V002",
        "libelle":"Porsche 964 Turbo",
        "vitesse_maxi":300
    },
    "voiture3":{
        "code":"V003",
        "libelle":"Ferrari 430",
        "vitesse_maxi":320
    }
}
```

Le nom de ce fichier, placé sur le site Web dans le même répertoire que le script JavaScript, est `voitures.json`.

Étudions maintenant pas à pas le script HTML/JavaScript.

Dans la section HTML `<body>`, nous trouvons essentiellement ceci :

```
<!-- Division d'affichage du résultat -->
<div id="divisionResultat"></div>

<!-- Appel de la fonction JSON ajaxJSON -->
<script type="text/JavaScript">
        ajaxJSON();
</script>
```

Une division HTML identifiée par `divisionResultat` est prévue pour accueillir les données extraites du fichier JSON `voitures.json`.

Une fonction nommée `ajaxJSON` est ensuite appelée. Le code aurait très bien pu être placé dans la section `<body>`. Le choix a été fait de positionner cette fonction dans la section HTML `<head>`.

Passons maintenant à l'essentiel, l'étude de la fonction `ajaxJSON`.

Le code débute par l'association d'une variable `resultat` à la division d'affichage `divisionResultat` évoquée précédemment :

```
/* Association de la variable resultat
à la division d'affichage divisionResultat */
var resultat = document.getElementById("divisionResultat");
```

Comme dans les exemples du chapitre précédent, un objet `XMLHttpRequest` va être employé :

```
/* Instanciation d'un objet de type XMLHttpRequest
/* NB : XMLHttpRequest est un objet ActiveX
ou JavaScript qui permet d'obtenir des données
au format XML, JSON, mais aussi HTML ou encore texte
simple à l'aide de requêtes HTTP. */
if (window.XMLHttpRequest) {
    // Code pour IE7+, Firefox, Chrome, Opera, Safari
    httpRequest = new XMLHttpRequest();
} else {
    // Code pour IE6, IE5
    httpRequest = new ActiveXObject("Microsoft.XMLHTTP");
}
```

Pour les versions anciennes du navigateur Microsoft Internet Explorer (versions 6 et antérieures), un objet ActiveX est utilisé en lieu et place de XMLHttpRequest.

Passons ensuite à l'ouverture du fichier `voitures.json` et à la définition de son type :

```
/* Ouverture du fichier voitures.json */
/* true : mode asynchrone -> le flux doit être
disponible entièrement avant son traitement */
httpRequest.open("GET", "voitures.json", true);

/* Définition du type de flux */
httpRequest.setRequestHeader("Content-type", "application/json");
```

Le paramètre `true` de la méthode `open` sert à signaler que le flux doit être totalement récupéré avant qu'il ne soit traité par l'application cliente. La méthode GET est utilisée car l'application cliente ne soumet pas de paramètre. Nous verrons ultérieurement un exemple de recherche de données dans une table MySQL avec une sélection dans le script client, nous utiliserons à cette occasion la méthode POST.

La méthode `setRequestHeader` précise que notre flux de données est au format JSON.

Nous arrivons maintenant au traitement du flux quand il est totalement disponible. La séquence de code est présentée ci-après dans son intégralité (une explication détaillée sera ensuite proposée) :

```
/* Traitement effectué dès que le flux est disponible */
```

```
httpRequest.onreadystatechange = function() {

    /* Test si requête terminée et test status OK */
    if (httpRequest.readyState == 4 && httpRequest.status == 200)
    {

            /* Affichages de contrôle */
            // alert("readystate : "+ httpRequest.readyState);
            // alert("status : "+ httpRequest.status);
            // alert("responseText : " + httpRequest.responseText);

            /* Conversion du flux JSON en objets JavaScript */
            var donneesJSON = JSON.parse(httpRequest.responseText);

            /* Initialisation de la variable resultat */
            resultat.innerHTML = "";

            /* Parcours des objets JavaScript */
            for (var obj in donneesJSON)
            {
                   /* Concaténation du résultat (une ligne par
                   enregistrement JSON) + un trait de séparation */
                   resultat.innerHTML += data[obj].libelle + " -> " +
                   data[obj].vitesse_maxi + " Km/h <hr />";
                   }

            }

    }

/* Pas d'envoi de données au travers de la requête XMLHttpRequest */
httpRequest.send(null);

/* Message affiché en attente du traitement du fichier voitures.json */
results.innerHTML = "Attente de traitement JSON ...";
```

Comme nous l'avions déjà vu dans le cadre du traitement des flux XML dans le chapitre précédent, des tests sont à effectuer sur les propriétés `readyState` et `status` de l'objet `XMLHttpRequest` (nommé `httpRequest` dans notre cas).

Ensuite des affichages de contrôle sont prévus (mise en commentaires dans le script ci-avant). À titre indicatif l'instruction :

```
alert("responseText : " + httpRequest.responseText);
```

donnerait l'affichage suivant :

```
responseText : {
    "voiture1":{
        "code":"V001",
        "libelle":"Porsche 930 Turbo",
        "vitesse_maxi":290
```

```
    },
    "voiture2":{
        "code":"V002",
        "libelle":"Porsche 964 Turbo",
        "vitesse_maxi":300
    },
    "voiture3":{
        "code":"V003",
        "libelle":"Ferrari 430",
        "vitesse_maxi":320
    }
}
```

Une fois le flux JSON récupéré, il faut le convertir en objets JavaScript. Cette opération fastidieuse peut être réalisée par une méthode native de JavaScript, `JSON.parse`. Il suffit de lui passer en paramètre le flux, c'est-à-dire `httpRequest.responseText`.

```
/* Conversion du flux JSON en objets JavaScript */
var donneesJSON = JSON.parse(httpRequest.responseText);
```

Il reste maintenant à réinitialiser (effacer) le contenu de la division d'affichage et à explorer les objets JavaScript, via une boucle `for` :

```
/* Initialisation de la variable resultat */
resultat.innerHTML = "";

/* Parcours des objets JavaScript */
for (var objet in donneesJSON)
{
    /* Concaténation du résultat */
    /* NB : Une ligne par enregistrement JSON
    + un trait de séparation */
    resultat.innerHTML += donneesJSON[objet].libelle + " -> "
    + donneesJSON[objet].vitesse_maxi + " Km/h <hr />";
}
```

Le script de la fonction `ajaxJSON` se termine par :

```
/* Pas d'envoi de paramètres au travers
de la requête XMLHttpRequest */
httpRequest.send(null);

/* Message affiché en attente du traitement
du fichier voitures.json */
results.innerHTML = "Attente de traitement JSON ...";
```

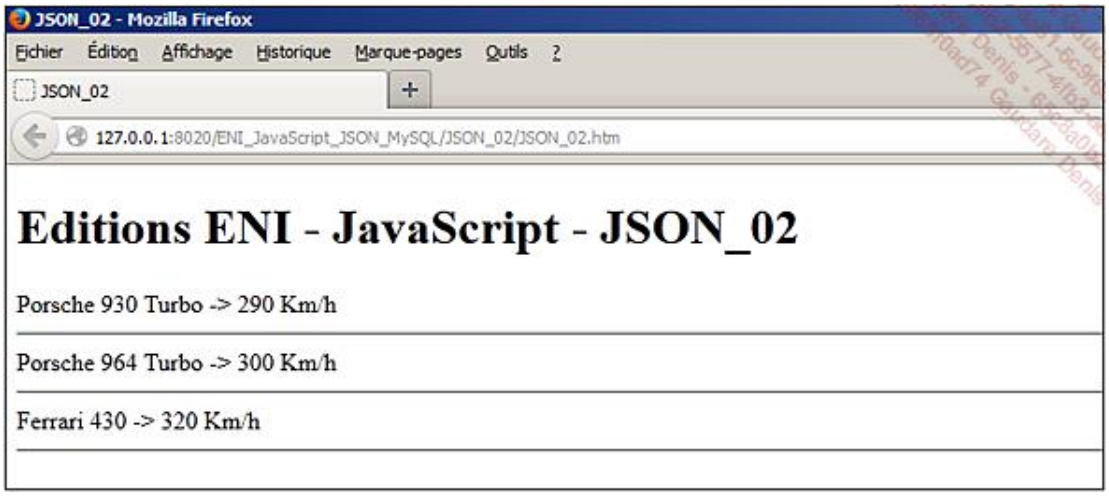
La méthode `send` de l'objet `httpRequest` a pour paramètre `null` qui signifie qu'aucune information n'est transmise au serveur par le client. Nous verrons ultérieurement un exemple de type POST avec justement transmission d'un paramètre. Dans notre cas (gestion du flux JSON en mode asynchrone) la méthode `send` doit être

placée en fin de script.

La dernière ligne du script est facultative. Elle permet de faire patienter l'utilisateur de l'application durant le traitement du flux JSON qui peut être long et complexe.

Compte rendu d'exécution

L'exécution du script `JSON_02.htm` donne ceci dans notre navigateur habituel (Mozilla Firefox) :



3. Exemple 3 : Lecture d'un fichier JSON via XMLHttpRequest et un script serveur en PHP

Commentaire des scripts JSON_03.htm et serveurJSON.php

Le troisième exemple présente de fortes similitudes avec l'exemple 2.

Le script client est presque identique, il a tout simplement été renommé `JSON_03.htm`.

Le fichier des données JSON contenant des descriptions de voitures est également repris (sans être renommé).

La différence dans ce nouvel exemple est que le flux JSON est lu par une application serveur écrite en PHP.

Voyons dans le détail les quelques modifications par rapport à l'application précédente.

Dans le script `JSON_02.htm` (exemple 2), nous avions cette ouverture de fichier :

```
/* Ouverture du fichier voitures.json */
/* true : mode asynchrone -> le flux doit être
disponible entièrement avant son traitement */
httpRequest.open("GET", "voitures.json", true);
```

Dans le script `JSON_03.htm`, il est fait référence au script serveur PHP de nom `serveurJSON.php` :

```
/* Ouverture du fichier voitures.json
via le script PHP serveurJSON */
```

```
/* true : mode asynchrone -> le flux doit être
disponible entièrement avant son traitement */
httpRequest.open("GET", "serveurJSON.php", true);
```

Étudions enfin le contenu du script serveur `serveurJSON.php` :

```
<?php

    // Type du flux
    header("Content-Type: application/json");

    // Lecture du fichier JSON voitures.json
    $donneesJSON = file_get_contents("voitures.json");

    // Envoi du flux JSON à l'application cliente
    echo $donneesJSON;

?>
```

Après avoir défini le type du flux (paramétrage de type `header`), le fichier JSON `voitures.json` est lu et est affecté sans transformation à une variable texte nommée `donneesJSON`.

Cette variable est finalement renvoyée à l'application cliente (`JSON_03.htm`).

Nous avons vu dans le script client `JSON_02.htm` (la démarche est identique pour `JSON_03.htm`) comment ce contenu est ensuite pris en compte.

À ce niveau de l'exposé des exemples, vous pourriez vous poser la question de l'intérêt de sous-traiter la lecture du fichier JSON `voitures.json` à un script serveur écrit en PHP. Nous allons voir dans les exemples suivants (mais vous le savez déjà depuis le chapitre précédent) comment utiliser le script serveur PHP pour extraire des données depuis une table de données MySQL et pour formater l'extraction au format JSON.

Compte rendu d'exécution

Avec les scripts de cet exemple publiés sur un espace Web accessible via l'URL http://christian.vigouroux.online.fr/js_json/JSON_03/, l'affichage obtenu à l'exécution du script `JSON_03.htm` est le suivant :

4. Exemple 4 : Lecture d'une table MySQL via XMLHttpRequest (serveur PHP et flux JSON)

Commentaire des scripts JSON_04.htm et serveurJSON.php

Vous le pressentez déjà, l'objectif de ce quatrième exemple va être de récupérer les données (trois voitures) stockées dans une table MySQL (déjà utilisée dans le cadre du précédent chapitre).

Alors que dans l'exemple 1 du chapitre précédent, nous avions privilégié les technologies SOAP et XML, cette fois-ci les données seront rapatriées au format JSON et traitées en conséquence par le script client `JSON_04.htm`.

Commençons par le décryptage du script serveur écrit en PHP, nommé `serveurJSON.php`.

Script serveur PHP serveurJSON.php

Le code source de ce script étant relativement nouveau, bien que présentant quelques similitudes avec celui vu dans l'exemple 1 du chapitre précédent, il est reproduit intégralement ci-après (il sera ensuite commenté dans le détail) :

```
<?php

    // Définition du header
    header("Content-Type: application/json");

    // Définition de la requête SQL à soumettre
    // à la base de données MySQL
    $requete_sql = "select
                code_voiture,
                libelle_voiture,
                vitesse_maximale_voiture
                from voitures
                order by code_voiture;";
    // return "Requête SQL : $requete_sql";

    // Paramètres SGBD MySQL
    $serveur_mysql = "localhost";
```

```
$utilisateur_mysql = "root";
$mot_de_passe_mysql = "";
$bdd_mysql = "json";

// Test de connexion à MySQL
if (
        (
        $connexion_mysql = @mysql_connect(
        $serveur_mysql,
        $utilisateur_mysql,
        $mot_de_passe_mysql)
        )
        === FALSE
)
{

    // Message d’erreur envoyé au client
    echo "";

}
else
{

    // Test accès à la base de données
    if ((@mysql_select_db($bdd_mysql, $connexion_mysql)) === FALSE)
    {

        // Message d’erreur envoyé au client
        echo "";

    }
    else
    {

        // Soumission de la requête SQL au moteur SQL de MySQL
        $resultat_sql
        = @mysql_query($requete_sql, $connexion_mysql);

        // Test du nombre d’enregistrements sélectionnés
        if (@mysql_num_rows($resultat_sql) <1)
        {

            // Message d’erreur envoyé au client
            // si pas d’enregistrement
            echo "";

        }
        else
        {

            while
            ($enregistrement = mysql_fetch_assoc($resultat_sql))
            {
                // Placement des 3 champs (code_voiture,
                // libelle_voiture et vitesse_maximale_voiture)
```

```
                    // de l'enregistrement en cours
                    // dans le tableau $lignes (qui contiendra
                    // au final l'intégralité des données)
                    $lignes[] = $enregistrement;
                }

                // Encodage en format JSON du tableau $lignes
                $donneesJSON = json_encode($lignes);

                // Envoi du résultat au client
                echo $donneesJSON;

            }

        }

    }

    // Fermeture de la connexion MySQL
    @mysql_close($connexion_mysql);

?>
```

En début de script, vous retrouvez le paramétrage du `header` déjà rencontré dans les précédents exemples :

```
// Définition du header
header("Content-Type: application/json");
```

La requête SQL paramétrée ensuite permettra l'extraction des trois enregistrements de la table MySQL `voitures` en les ordonnant sur les valeurs du champ `code_voiture` :

```
$requete_sql = "select
            code_voiture,
            libelle_voiture,
            vitesse_maximale_voiture
            from voitures
            order by code_voiture;";
```

Viennent ensuite les paramètres de connexion au système de gestion de base de données MySQL et à la base de données de nom `json`. Vous noterez que notre exemple a été déployé sur un serveur local (EasyPHP en l'occurrence) pour pouvoir disposer d'une version du langage PHP (5.2 minimum) supportant la fonction `json_encode`. Ces paramètres de connexion sont :

```
// Paramètres SGBD MySQL
$serveur_mysql = "localhost";
$utilisateur_mysql = "root";
$mot_de_passe_mysql = "";
$bdd_mysql = "json";
```

Comme dans l'exemple 1 du chapitre précédent, des tests sont effectués en matière de connexion au SGBD et à la base de données. Dans notre script minimaliste, les messages d'erreurs à renvoyer à l'application client n'ont pas été formalisés, un simple `echo "";` est prévu en cas d'erreur.

Attardons-nous sur l'essentiel, la récupération des données depuis la table MySQL `voitures` et l'encodage en un flux JSON et le renvoi du flux vers l'application cliente :

```
while
($enregistrement = mysql_fetch_assoc($resultat_sql))
{
    // Placement des 3 champs (code_voiture,
    // libelle_voiture et vitesse_maximale_voiture)
    // de l'enregistrement en cours
    // dans le tableau $lignes (qui contiendra
    // au final l'intégralité des données)
    $lignes[] = $enregistrement;
}

// Encodage en format JSON du tableau $lignes
$donneesJSON = json_encode($lignes);

// Envoi du résultat au client
echo $donneesJSON;
```

La structure itérative `while` balaie l'ensemble des enregistrements correspondant à la requête SQL (`$requete_sql`). Pour chaque enregistrement lu, les valeurs des champs sont placées dans un tableau de nom `$enregistrement`. Ce tableau est lui-même affecté à un autre tableau (qui devient de la sorte une espèce de tableau à deux dimensions) de nom `$lignes`.

Ensuite par la fonction PHP `json_encode`, disponible dans les versions récentes de PHP, le tableau `$lignes` sera transformé en un flux JSON.

Enfin le flux JSON est envoyé à l'application cliente.

Script client HTML/JavaScript JSON_04.htm

Ce script est très proche de l'exemple 3 de ce même chapitre. Seul l'affichage des données a été modifié légèrement :

```
/* Parcours des objets JavaScript */
for (var objet in donneesJSON)
{
    /* Concaténation du résultat */
    resultat.innerHTML += "Code voiture : "
    + donneesJSON[objet].code_voiture + "<br />";
    resultat.innerHTML += "Libellé : "
    + donneesJSON[objet].libelle_voiture + "<br />";
    resultat.innerHTML += "Vitesse maximale : "
    + donneesJSON[objet].vitesse_maximale_voiture + "<hr />";
}
```

À l'exécution de ce script le flux JSON récupéré est :

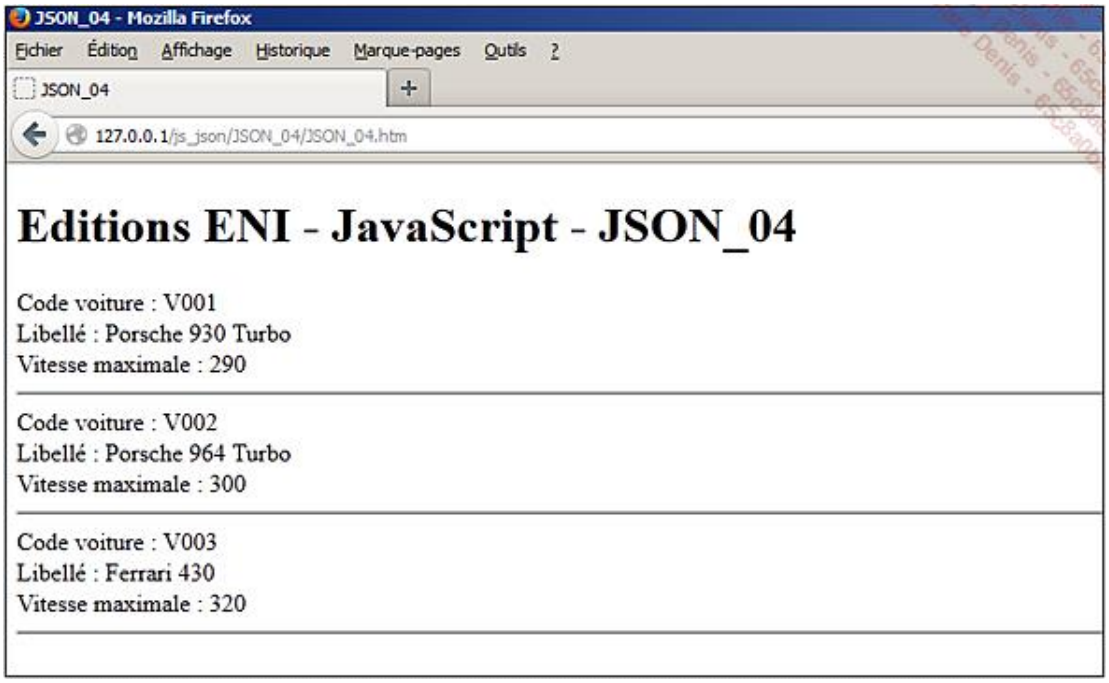
```
[{"code_voiture":"V001","libelle_voiture":"Porsche 930 Turbo",
"vitesse_maximale_voiture":"290"},
{"code_voiture":"V002","libelle_voiture":"Porsche 964 Turbo",
"vitesse_maximale_voiture":"300"},
{"code_voiture":"V003","libelle_voiture":"Ferrari 430",
"vitesse_maximale_voiture":"320"}]
```

Dans le cas où le module PHP de votre serveur Web (Apache dans la majorité des cas) ne supporte pas la fonction `json_encode`, la partie n'est tout de même pas perdue. Il vous faut via du code PHP construire le flux JSON.

Compte rendu d'exécution

L'hébergeur Free (utilisé pour les exemples du chapitre précédent) ne proposant pas la fonction `json_encode`, les scripts de cet exemple ont été déployés dans un intranet de type EasyPHP (version 12.1).

Via l'URL http://127.0.0.1/js_json/JSON_04/JSON_04.htm, nous obtenons ceci :



Outre l'installation d'EasyPHP, il faudra, pour exécuter cet exemple sur votre configuration personnelle, créer une base de données MySQL de nom json et y déployer la table voitures.

5. Exemple 5 : Recodage de l'exemple 4 avec une liste déroulante

Ce cinquième exemple va être assez proche de l'exemple 2 du chapitre précédent. En clair, l'opérateur pourra au travers d'une liste déroulante choisir la voiture de sport dont il souhaite obtenir les caractéristiques.

Cette possibilité a quelques impacts à la fois sur le script client (`JSON_05.htm`) et sur le script serveur (`serveurJSON.php`).

Commençons cette fois-ci par le décryptage du script client.

Script client HTML/JavaScript JSON_05.htm

Dans la section HTML <body> vous trouvez la liste déroulante et la division d'affichage du résultat :

```
<!-- Formulaire de saisie du code de la voiture à interroger -->
<form name="formulaire">
    <!-- Liste déroulante des choix -->
    Code de la voiture :
    <select
    id="liste"
    onchange="controlerChoixListe(document.getElementById('liste'),
    'Veuillez choisir un code')">
        <option value="CODE" selected> Code voiture </option>
        <option value="V001"> V001 </option>
        <option value="V002"> V002 </option>
        <option value="V003"> V003 </option>
    </select>
</form>

<!-- Division d'affichage du résultat -->
<div id="divisionResultat"></div>
```

Outre l'habituelle fonction ajaxJSON, la section HTML <head> intègre aussi une fonction de gestion de la liste déroulante, controlerChoixListe. Cette fonction est quasiment identique à celle de l'exemple 2 du chapitre précédent :

```
/* Fonction testant si un choix a été fait
dans la liste déroulante */
function controlerChoixListe(liste, messageAlerte) {
    if (liste.value == "CODE") {
        /* Affichage d'un message d'alerte */
        alert(messageAlerte);
        /* Focus sur le champ en erreur */
        liste.focus();
        /* Valeur de retour */
        return false;
    } else {
        /* Appel de la fonction ajaxJSON */
        ajaxJSON();
        /* Valeur de retour */
        return true;
    }
}
```

La seule différence est bien évidemment le nom de la fonction appelée, c'est-à-dire ajaxJSON :

```
            /* Appel de la fonction ajaxJSON */
```

```
            ajaxJSON();
```

Attardons-nous aussi un peu sur la fonction `ajaxJSON`.

Il y a quelques différences par rapport au script de l'exemple 1 de ce même chapitre :

```
/* Ouverture du fichier voitures.json
via le script PHP serveurJSON.php*/
/* true : mode asynchrone -> le flux doit être
disponible entièrement avant son traitement */
httpRequest.open("POST", "serveurJSON.php", true);
```

La méthode `open` est utilisée avec le paramètre POST (en lieu et place de GET) dans la mesure où un paramètre va être posté au script `serveurJSON.php`.

Avec le paramètre POST l'envoi des paramètres se fait au travers du corps de la requête alors qu'avec le paramètre GET l'envoi des paramètres se fait dans l'URL elle-même (sous la forme : http://monsite/monscriptserveur?param1=valeur1&param2=valeur2 par exemple)

Le type de flux vers le serveur est aussi différent :

```
/* Définition du type de flux vers le serveur */
httpRequest.setRequestHeader("Content-type",
"application/x-www-form-urlencoded");
```

Vous noterez que `application/x-www-form-urlencoded` est la valeur à retenir pour l'envoi de paramètres depuis un formulaire HTML (liste déroulante dans notre cas). Les valeurs de paramètres seront encodées (remplacement des espaces à titre d'exemple par `%20`).

Vous trouverez une explication détaillée sur le sujet à l'adresse suivante : http://assiste.com.free.fr/p/faq_webmaster/HTML_ASCII_Escape_codes_H2_%20!$%25&%27%28%29+,-..html

Il reste à voir comment expédier le paramètre (code de la voiture) au serveur via la méthode `send` de l'objet `httpRequest` :

```
/* Envoi de données au travers de la requête XMLHttpRequest
au script PHP serveurJSON.php */
var parametre = 'code=';
parametre += document.getElementById('liste').value;
// alert("Paramètre : " + parametre);
httpRequest.send(parametre);
```

Une variable `parametre` est initialisée avec la séquence `code=` suivie du choix effectué dans la liste déroulante (`document.getElementById('liste').value`). Ce paramètre est ensuite envoyé au serveur via :

```
httpRequest.send(parametre);
```

Script serveur PHP serveurJSON.php

Le script serveur est assez proche de celui vu dans l'exemple 1.

Voyons les quelques modifications apportées :

```
// Récupération du paramètre passé par l'application cliente
$code = $_POST["code"];

// Définition de la requête SQL à soumettre
// à la base de données MySQL
$requete_sql =
"select code_voiture, libelle_voiture,
vitesse_maximale_voiture
from voitures
where code_voiture='$code';";
```

Une variable `$code` récupère la valeur du paramètre code passé par le script client. Les valeurs possibles sont dans notre cas V001, V002 et V003.

La requête SQL a été aménagée avec le rajout d'une clause SQL de sélection portant justement sur ce code :

```
where code_voiture='$code'
```

Compte rendu d'exécution

Notre hébergeur habituel (Free), ne proposant pas la fonction `json_encode`, les scripts de cet exemple ont été déployés dans un intranet de type EasyPHP (version 12.1).

Via l'URL http://127.0.0.1/js_json/JSON_05/JSON_05.htm, nous obtenons ceci :