

Mise en œuvre du stockage distant au travers d'exemples

Deux exemples complets de mise en œuvre vont vous être présentés.

Pour les besoins de ces exemples sera utilisée une table de nom `voitures` comportant trois champs :

- `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 !).

Dans le cadre de vos expérimentations personnelles, vous ne pourrez pas utiliser la base de données support des exemples du livre. L'identifiant et le mot de passe sont d'ailleurs masqués dans les reproductions des codes "source" qui vont suivre.

Il vous faudra donc disposer d'un SGBD MySQL auquel vous aurez accès. L'obtention ou la réservation d'un accès à un tel type de serveur sort du cadre de notre exposé dans ce livre.


Pour que vous puissiez tester les exemples, il vous faudra donc déployer une table de nom `voitures` sur votre serveur MySQL et bien sûr renseigner l'identifiant et le mot de passe connexion dans les scripts.

Pour faciliter le déploiement, un script SQL de création de la table `voitures` est fourni ci-après. Il vous faudra l'exécuter dans la fenêtre SQL de votre interface d'administration MySQL (PHPMyAdmin en règle générale).

```
CREATE TABLE voitures(  
    code_voiture CHAR(4),  
    libelle_voiture CHAR(20),  
    vitesse_maximale_voiture INTEGER  
);
```

Un jeu d'essai doit aussi être mis en place pour l'exploitation des exemples. Vous pourrez exécuter le script ci-dessous pour peupler votre table `voitures` :

```
INSERT INTO voitures (code_voiture, libelle_voiture,  
vitesse_maximale_voiture)  
VALUES ("V001", "Porsche 930 Turbo", 290);  
INSERT INTO voitures (code_voiture, libelle_voiture,  
vitesse_maximale_voiture)  
VALUES ("V002", "Porsche 964 Turbo", 300);  
INSERT INTO voitures (code_voiture, libelle_voiture,  
vitesse_maximale_voiture)  
VALUES ("V003", "Ferrari 430", 320);
```

 Les deux scripts SQL ainsi que tous les scripts des exemples de ce livre sont téléchargeables gratuitement sur la page Informations générales.

1. Exemple 1 : Accès Ajax sur BDD MySQL (liste de l'ensemble des voitures)

Dans un premier exemple, le listage de tous les enregistrements d'une table MySQL de nom `voitures` est prévu.

Ce premier exemple utilise essentiellement quatre éléments :

- un script HTML (contenant bien sûr également des séquences de code JavaScript) de nom `client_ws_01.htm`,
- un script PHP de nom `serveur_ws_01.php`,
- un script PHP `nusoap.php`,
- une table MySQL de nom `voitures` (évoquée précédemment).

Pour pouvoir utiliser ces scripts dans le cadre d'une expérimentation personnelle, les fichiers `client_ws_01.htm`, `serveur_ws_01.php` et `nusoap.php` devront être placés par votre client FTP habituel (Mozilla Filezilla par exemple) dans un répertoire (par exemple de nom `js_nusoap`) sur votre serveur Web.

Une fois de plus l'obtention ou la réservation d'un accès d'un espace de publication Web sort du cadre de notre exposé dans ce livre.

La première mini-application qui va être présentée est aussi un bon exemple de ce que l'on appelle communément Ajax (*Asynchronous JavaScript and XML*).

Ajax regroupe un ensemble de technologies (JavaScript, XML, CSS, XML, DOM et XMLHttpRequest) et permet de bâtir des applications Web dynamiques utilisant en particulier des bases de données et présentant des "interfaces utilisateur" enrichies (RIA).

DOM et JavaScript servent à modifier l'information présentée dans le navigateur.

L'objet XMLHttpRequest est utilisé pour assurer les échanges de données avec le serveur Web et ceci en mode asynchrone.

XML structure les flux de données entre le serveur Web et le navigateur. D'autres technologies, comme JSON, peuvent être utilisées en lieu et place du XML pour coder les flux de données entre le serveur Web et le navigateur.

Ajax est normalement compatible avec les principaux navigateurs comme Microsoft Internet Explorer, Mozilla Firefox, Google Chrome, Konqueror ou encore Opera.

Dans notre exemple justement, la grande majorité des technologies rassemblées sous le vocable Ajax vont être mises en œuvre :

- un script client (HTML + JavaScript) préparera un message SOAP/XMLHttpRequest à l'adresse d'une application serveur écrite en PHP,
- le script serveur accédera à une base de données hébergée dans un SGBD MySQL,
- via une librairie dédiée (NuSOAP), la réponse (les enregistrements correspondants à une extraction SQL), c'est-à-dire un flux XML sera renvoyé à l'application cliente,
- l'application cliente, via des méthodes JavaScript de traitement du DOM, récupérera les données pour affichage.

Cet exemple basé sur Ajax est donc relativement complet, il n'y manque que la prise en compte du CSS (mise en forme, par une feuille de style, des données récupérées par le script client) et de JSON (dans notre cas le format XML a été privilégié pour la structure du message en retour du serveur). Nous aurons l'occasion de découvrir le format JSON dans une série d'exemples présentés dans le cadre du prochain chapitre.



SOAP est un protocole orienté objet bâti sur XML qui permet la transmission de messages entre objets applicatifs distants. Un objet peut, via ce protocole, invoquer des méthodes d'objets physiquement situés sur d'autres serveurs. Le transfert des messages SOAP se fait le plus couramment par l'intermédiaire du protocole HTTP.

Passons maintenant à l'étude détaillée de ces différents scripts.

Script nusoap.php

Vous pourrez récupérer l'intégralité de cette librairie PHP par téléchargement à partir du site <http://sourceforge.net/projects/nusoap/>.

Pour les besoins de nos deux exemples, seul le script `nusoap.php` est nécessaire. Il faudra donc soit l'extraire du package (fichier zippé) récupéré ou le rechercher dans les scripts d'accompagnement de ce livre mis à votre disposition sur la page Informations générales.

Le script `nusoap.php` devra être positionné dans le même répertoire que les scripts `client_ws_01.htm` et `serveur_ws_01.php` (par exemple `js_nusoap`).

Le commentaire du contenu du fichier `nusoap.php` n'est pas prévu dans le cadre de ce livre. Sachez toutefois que ce fichier de près de 6000 lignes de code contient plus d'une centaine de fonctions.

Script HTML client_ws_01.htm

Dans ce script vous trouverez dans la section HTML `<body> . . . </body>` un appel à une fonction JavaScript de nom `appelerServeurDistant`.

```
<!-- Début script JavaScript -->
<script type="text/JavaScript">

    /* Affichage du nom du script */
    alert("WS_01");

    /* Appel de la fonction appelerServeurDistant
    listant les données de la table MySQL distante voitures */
    appelerServeurDistant();

</script>
```

Sous cet appel à la fonction `appelerServeurDistant` est placée une division de nom `resultat` qui sera renseignée une fois les données récupérées depuis le serveur de données MySQL.

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

Passons maintenant à l'essentiel, l'étude détaillée des fonctions JavaScript assurant la récupération des données et leur traitement.

Fonction appelerServeurDistant

La première fonction, nommée `appelerServeurDistant` a pour objectifs principaux :

- la préparation d'une requête SOAP à transmettre au serveur distant (serveur Apache en ce qui nous concerne).

- l'attente du traitement par le serveur,
- la récupération de la réponse du serveur distant (réponse au format XML dans notre cas),
- la désérialisation de la réponse XML,
- l'affichage des données dans la division d'affichage resultat.

Le code complet de cette fonction est présenté ci-après :

```

/* Fonction appelerServeurDistant */
function appelerServeurDistant()
{
    /* Message SOAP à transmettre au serveur */
    messageSOAP = '<?xml version="1.0" encoding="ISO-8859-1"?>';
    messageSOAP += '<SOAP-ENV:Envelope SOAP-ENV:encodingStyle='
    "http://schemas.xmlsoap.org/soap/encoding/"';
    messageSOAP += ' xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"';
    messageSOAP += ' xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"';
    messageSOAP += ' xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:si="http://soapinterop.org/xsd">';
    messageSOAP += '<SOAP-ENV:Body>';
    messageSOAP += '<ns1:listeVoitures xmlns:ns1='
    "http://christian.vigouroux.online.fr"></ns1:listeVoitures>';
    messageSOAP += '</SOAP-ENV:Body>';
    messageSOAP += '</SOAP-ENV:Envelope>';
    // alert("messageSOAP : " + messageSOAP);

    /*
    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");
    }

    /* Accès au serveur SOAP */
    httpRequest.open("POST", url, true);

    /*
    Application de la méthode overrideMimeType
    pour signaler une réponse du serveur SOAP
    en format texte ou XML

```

```

    */
    if (httpRequest.overrideMimeType)
    {
        httpRequest.overrideMimeType("text/xml");
    }

    /*
    Préparation de l'en-tête du message SOAP
    (Content-Type positionné à text/xml)
    NB : Cf. https://en.wikipedia.org/wiki/List\_of\_HTTP\_header\_fields
    pour les autres paramètres possibles
    */
    httpRequest.setRequestHeader("Content-Type", "text/xml");

    /*
    Envoi de la requête SOAP au serveur
    avec un timeout de 60000 ms (soit 60 secondes)
    NB : La requête sera en erreur une fois le délai atteint
    */
    httpRequest.send(messageSOAP);
    valTimeout=setTimeout("timeout(httpRequest);", 60000);

    /*
    Parsage et copie de la réponse dans la division HTML resultat
    dès que celle-ci a été fournie par le serveur SOAP
    */
    httpRequest.onreadystatechange=parserDOM;
}

```

La première partie (la plus complexe) consiste à préparer le message SOAP à transmettre au serveur distant.

Sans être exhaustif sur les différents éléments à intégrer dans cette enveloppe SOAP (terme utilisé pour désigner le message), notons les deux points sur lesquels vous devrez intervenir dans le cas d'une réutilisation du code de cette fonction dans vos développements personnels.

Tout d'abord, dans le script présenté, `listervoitures` est une méthode intégrée au script `serveur_ws_01.php` que nous étudierons plus tard.

```

messageSOAP += '<ns1:listervoitures xmlns:ns1=
"http://christian.vigouroux.online.fr"></ns1:listervoitures>';

```

Ensuite, bien que ce ne soit pas absolument obligatoire, vous pouvez aussi personnaliser le nom de domaine dans cette même ligne de code :

```

xmlns:ns1="http://christian.vigouroux.online.fr"

```

Pour le reste des éléments constituant l'enveloppe SOAP, nous vous renvoyons sur le site de partage de connaissances Wikipedia (<http://fr.wikipedia.org/wiki/SOAP>) ou encore sur la page en français du W3C sur le sujet (<http://www.w3.org/2002/07/soap-translation/soap12-part0.html>).

Une fois l'enveloppe SOAP préparée, il convient d'instancier un objet `XMLHttpRequest` qui permettra de

communiquer avec le serveur distant.

Une fois de plus pour obtenir une explication plus complète sur la notion de `XMLHttpRequest`, vous pourrez vous reporter à Wikipedia (<http://fr.wikipedia.org/wiki/XMLHttpRequest>).

En résumé, `XMLHttpRequest` est un objet ActiveX ou JavaScript qui permet d'obtenir des données au format XML, JSON, mais aussi HTML, ou encore du texte simple à l'aide de requêtes HTTP. Pour des navigateurs d'ancienne génération (Microsoft Internet Explorer version 6 ou versions antérieures), il faudra avoir recours à un objet ActiveX. Un test doit donc être effectué pour déterminer si le navigateur requiert un ActiveX ou non (dans ce cas, un objet JavaScript sera 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");
}
```

Ensuite une requête de connexion est émise au serveur :

```
/* Accès au serveur SOAP */
httpRequest.open("POST", url, true);
```

La méthode `open` de l'objet `httpRequest` instancié précédemment est utilisée avec trois paramètres :


- `POST`
- `url`
- `true`

Le paramètre `POST` sert à signaler que l'URL passée au serveur est sans paramètres (dans le cas contraire, le paramètre `GET` serait employé). Il faut signaler de toute façon qu'en matière de sécurité le passage de paramètres à un serveur via l'URL est une technique peu fiable.

Le paramètre `url` contient l'URL du script serveur sollicité (`serveur_ws_01.php` dans notre cas). La valeur a été paramétrée en début de script :

```
/* Variables globales à toutes les fonctions JavaScript */
var httpRequest = null;
```

```
var valTimeout = null;
var url =
"http://christian.vigouroux.online.fr/js_nusoap/serveur_ws_01.php";
var messageSOAP = null;
```

 Vous trouverez des références intéressantes sur la méthode `open` aux adresses suivantes :

https://developper.mozilla.org/fr/docs/AJAX/Premiers_pas et http://openweb.eu.org/articles/objet_xmlhttprequest


Le troisième paramètre précise que notre requête est asynchrone. S'il est positionné à `true`, l'exécution de la fonction JavaScript se poursuivra en attendant l'arrivée de la réponse du serveur. C'est le premier A d'AJAX.

La séquence de code suivante sert à préciser que la réponse attendue du serveur est de type texte ou XML (XML en réalité dans notre cas, comme nous le verrons plus loin) :

```
/*
Application de la méthode overrideMimeType
pour signaler une réponse du serveur SOAP
en format texte ou XML
*/
if (httpRequest.overrideMimeType)
{
    httpRequest.overrideMimeType("text/xml");
}
```

Il reste ensuite à rajouter à notre message SOAP une indication quant au `Content-Type` :

```
/*
Préparation de l'en-tête du message SOAP
(Content-Type positionné à text/xml)
NB : Cf. https://en.wikipedia.org/wiki/List_of_HTTP_header_fields
pour les autres paramètres possibles
*/
httpRequest.setRequestHeader("Content-Type", "text/xml");
```

 Comme indiqué dans le commentaire dans le script, vous trouverez si vous le souhaitez une explication détaillée sur la notion de `Content-Type` sur le site Wikipedia (https://en.wikipedia.org/wiki/List_of_HTTP_header_fields).

La requête peut maintenant être émise au serveur et une gestion de `timeout` peut aussi être faite :

```
/*
Envoi de la requête SOAP au serveur
avec un timeout de 60000 ms (soit 60 secondes)
NB : La requête sera en erreur une fois le délai atteint
*/
httpRequest.send(messageSOAP);
valTimeout = setTimeout("timeout(httpRequest);", 60000);
```

La variable `messageSOAP` est émise vers le serveur par l'intermédiaire de la méthode `send` de l'objet `httpRequest`. La méthode JavaScript `setTimeout` fixe une valeur de `timeout` à 60000 (1 minute). Notez que la méthode `timeout`, présentée ci-après, interrompra la requête une fois le délai atteint :

```
/* Fonction timeout */
function timeout(ajaxOBJ)
{
    /* Interruption du traitement */
    ajaxOBJ.abort();
}
```

Cette fonction se termine par l'appel à la fonction `parserDOM` comme ceci :

```
/*
Parsage et copie de la réponse dans la division HTML resultat
dès que celle-ci a été fournie par le serveur SOAP
*/
httpRequest.onreadystatechange=parserDOM;
```

Rappelons que l'événement `onreadystatechange` permet de déclencher un traitement (la fonction `parserDOM` dans notre cas) à chaque changement de valeur de l'indicateur `readyState` associé à l'objet `httpRequest`. À titre indicatif, les valeurs de cet indicateur peuvent être :

- 0 : La requête n'est pas initialisée,
- 1 : La connexion avec le serveur a été établie,
- 2 : La requête a été reçue par le serveur,
- 3 : le traitement est en cours sur le serveur,
- 4 : la requête est terminée.

Fonction parserDOM

Cette fonction est appelée par la fonction précédente (`appelerServeurDistant`).

Dans le code de cette fonction, nous allons mettre en œuvre pour la première fois dans ce livre une instruction particulière, le `try ... catch`.

Le `try` délimite un bloc d'instructions à tester et dans le bloc `catch` une réponse est prévue dans le cas où une exception (erreur) est levée dans le bloc `try`.

Le bloc `try` (toujours présent par définition) peut comporter de nombreuses instructions. Pour le traitement des exceptions, il peut y avoir plusieurs `catch` consécutifs (voire aucun). Il est possible de prévoir également en dernière position, après la série de `catch`, un dernier bloc nommé `finally` dans lequel seront intégrés des instructions systématiquement exécutées avant le déclenchement des instructions suivantes du programme.

Avant de passer à l'étude détaillée de la fonction `parserSOAP`, regardons un exemple complet de mise en œuvre de `try ... catch` :

Cet exemple commence par une fonction (verifJour) qui retourne le libellé de jour d'un numéro de jour passé en paramètre (ou un message d'erreur si le numéro est invalide) :

```
/* Définition de la fonction verifJour */
function verifJour(numeroJour)
{
    /*
     Décrémentation du paramètre passé pour tenir compte
     de l'indiciage des valeurs du tableau débutant à 0
    */
    numeroJour = numeroJour - 1;

    /*
     Déclaration d'un tableau des libellés
     de jour de la semaine
    */
    var tableauJours = new Array("Lundi", "Mardi", "Mercredi",
    "Jeudi", "Vendredi", "Samedi", "Dimanche");

    /* Test de validité du paramètre */
    if (tableauJours[numeroJour] != null)
    {
        /* Valeur de retour (paramètre valide) */
        return tableauJours[numeroJour];
    }
    else
    {
        /* Valeur de retour (paramètre invalide) */
        return "Erreur dans le n° de jour";
    }
}
```

Appelons maintenant cette fonction comme suit :

```
/* Affichage du libellé du jour dans la semaine */
var numJour = 8;
try
{
    /* Récupération du libellé du jour */
    alert(verifJour(numJour));
    alert(verificationJour(numJour))
}
catch(erreur)
{
    alert(erreur);
}
```

Une variable numJour est déclarée et initialisée à la valeur 8, ce qui devrait nous poser un problème car il n'y a que 7 jours dans la semaine.

Dans le bloc try, deux tentatives de récupération du libellé du jour de la semaine sont effectuées.

Dans le premier cas (`alert(verifJour(numJour));`) le résultat affiché sera le message " Erreur dans le n° de jour" prévu dans le `else` de la fonction `verifJour`.

Dans le second cas, une erreur de programmation a été commise, il est fait appel à une fonction inexistante de nom `verificationJour`. Le `catch` va donc remplir son rôle et afficher un message d'erreur lié à cette anomalie (`ReferenceError: verificationJour is not defined`).

Revenons maintenant à la fonction `parserDOM`, qui va intégrer justement un `try ... catch`. Le code source de cette fonction est reproduit intégralement ci-après :

```
/* Fonction parserDOM */
function parserDOM()
{
    try
    {
        /* Test si requête terminée et résultat reçu */
        if (httpRequest.readyState == 4)
        {
            /* Test status OK pour la requête http */
            if(httpRequest.status == 200)
            {
                /* Mise en place du timeout */
                clearTimeout(valTimeout);
                /* Récupération de la réponse */
                var text = httpRequest.responseText;
                // alert("text : " + text);
                /*
                Choix du parser pour décoder le flux XML envoyé
                par le serveur
                */
                if (window.DOMParser)
                {
                    /*
                    Parser pour IE7 et versions postérieures,
                    Firefox, Chrome, Opera, Safari
                    */
                    parser = new DOMParser();
                    /* Récupération du flux à parser */
                    xmlDoc = parser.parseFromString(text, "text/xml");
                }
                else
                {
                    /* Parser pour IE6 et versions antérieures */
                    xmlDoc = new ActiveXObject("Microsoft.XMLDOM");
                    /*
                    La récupération du flux devra être complète avant
                    le début du parsing
                    */
                    xmlDoc.async = "false";
                    /* Récupération du flux à parser */
                    xmlDoc.loadXML(text);
                }
            }
        }
    }
    /* Parsage */
}
```

```

/*
NB : Le serveur SOAP renvoie les valeurs lues dans la
table MySQL voitures dans un flux XML avec comme balise
d'encadrement <item> ... </item>
*/
var html = "";
// alert("Nombre de champs intégrés dans le flux XML : "
// + xmlDoc.getElementsByTagName("item").length);
for (i=0; i<xmlDoc.getElementsByTagName("item").length; i++)
{
    /*
    NB : childNodes[0] désigne le premier enfant de
    l'élément nommé item et nodeValue représente la
    valeur associée
    */
    if (i % 3 == 0)
    {
        html += "<br /><br />Code : " +
        xmlDoc.getElementsByTagName("item")[i]
        .childNodes[0].nodeValue;
    }
    if (i % 3 == 1)
    {
        html += "<br />Libellé : " +
        xmlDoc.getElementsByTagName("item")[i]
        .childNodes[0].nodeValue;
    }
    if (i % 3 == 2)
    {
        html += "<br />Vitesse maximale : " +
        xmlDoc.getElementsByTagName("item")[i]
        .childNodes[0].nodeValue;
    }
}
/*
Placement du résultat du parsing
dans la division HTML resultat
*/
var divisionResultat =
document.getElementById("resultat");
divisionResultat.innerHTML = html;
}
}
}
catch(e)
{
    /* Cas d'erreur */
    alert("Description de l'erreur : " + e.description);
}
}
}

```

Passons rapidement sur le mécanisme `try ... catch` qui a pour objectif de repérer des erreurs. Le code débute ensuite par deux tests imbriqués (ils auraient pu être regroupés) :

- un premier test vérifiant si la requête s'est achevée (`httpRequest.readyState == 4`),
- un second test vérifiant le statut de la réponse.

L'attribut `readyState` de l'objet `httpRequest` peut prendre les valeurs suivantes :

- 0 : Requête non initialisée,
- 1 : Requête en cours de chargement,
- 2 : Requête chargée,
- 3 : Requête en cours d'interaction,
- 4 : Requête terminée.

Seule la valeur 4 pour l'attribut `readyState` doit normalement être acceptée.

La seconde vérification porte sur le code d'état (`status`) de la réponse HTTP fournie par le serveur. Les valeurs possibles de ce code sont données sur le site du W3C (<http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>). Seul le cas d'un statut "200 OK." est accepté dans notre cas.

Ensuite un `timeout` est mis en place par l'intermédiaire de la méthode `clearTimeout`. Le paramètre `valTimeout` est une variable déclarée en début de script et initialisée à `null` :

```
/* Mise en place du timeout */
clearTimeout(valTimeout);
```

La séquence suivante est la récupération de la réponse renvoyée par le serveur :

```
/* Récupération de la réponse */
var text = httpRequest.responseText;
```

La réponse est stockée dans une variable `text` (déclarée pour l'occasion) par la récupération de la valeur de l'attribut `responseText` de l'objet `httpRequest`.

À titre indicatif (cela dépend bien entendu des données stockées dans la table MySQL voitures), à l'exécution la réponse du serveur est :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<SOAP-ENV:Envelope SOAP-ENV:encodingStyle=
"http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:si="http://soapinterop.org/xsd"><SOAP-ENV:Body>
<ns1:liesterVoituresResponse xmlns:ns1=
"http://christian.vigouroux.online.fr">
<return xsi:type="SOAP-ENC:Array" SOAP-ENC:arrayType="xsd:string[9]">
<item xsi:type="xsd:string">V001</item>
<item xsi:type="xsd:string">Porsche 930 Turbo</item>
```

```

<item xsi:type="xsd:string">290</item>
<item xsi:type="xsd:string">V002</item>
<item xsi:type="xsd:string">Porsche 964 Turbo</item>
<item xsi:type="xsd:string">300</item>
<item xsi:type="xsd:string">V003</item>
<item xsi:type="xsd:string">Ferrari 430</item>
<item xsi:type="xsd:string">320</item></return>
</nsl:listeVoituresResponse></SOAP-ENV:Body></SOAP-ENV:Envelope>

```

Dans ce flux XML en retour du serveur SOAP, vous aurez repéré plusieurs jeux de balises `<item> ... </item>` encadrant les valeurs extraites de la table `voitures`, soit dans notre cas :

- V001, Porsche 930 et 290 pour le premier enregistrement,
- V002, Porsche 964 et 300 pour le deuxième enregistrement,
- V003, Ferrari 430 et 320 pour le troisième enregistrement.

Le choix du nom de la balise a été imposé par la librairie NuSOAP dont nous parlerons plus loin.

Il serait bien sûr possible de repérer par un traitement de chaînes très sophistiqué les valeurs des champs. Par contre comme la réponse du serveur SOAP est formalisée en un flux XML, il va être possible d'automatiser la récupération des valeurs des champs par un dispositif dédié exploitant le modèle DOM de ce flux.

Pour cette opération de transformation, il faut tenir compte du navigateur. Pour les navigateurs récents et pour Microsoft Internet Explorer versions 7 et ultérieures, il sera fait recours à l'objet `DOMParser` nativement intégré. Pour les versions anciennes de Microsoft Internet Explorer, un objet ActiveX `Microsoft.XMLDOM` devra être utilisé.

```

/* Choix du parser pour décoder le flux XML envoyé par le serveur */
if (window.DOMParser)
{
    /* Parser pour IE7 et versions postérieures, Firefox, Chrome,
    Opera, Safari */
    parser = new DOMParser();
    /* Récupération du flux à parser */
    xmlDoc = parser.parseFromString(text, "text/xml");
}
else
{
    /* Parser pour IE6 et versions antérieures */
    xmlDoc = new ActiveXObject("Microsoft.XMLDOM");
    /* La récupération du flux devra être complète
    avant le début du parsing */
    xmlDoc.async = "false";
    /* Récupération du flux à parser */
    xmlDoc.loadXML(text);
}

```

Dans chacun des cas, le flux XML est stocké dans un objet nommé `xmlDoc` qui devra ensuite être parsé.

Il y a aussi une petite particularité pour les navigateurs anciens (IE6 et versions antérieures), le parsing ne doit pas débuter avant que le flux XML ne soit totalement disponible (`xmlDoc.async = "false";`).

Nous arrivons à une étape cruciale du traitement, le passage :

```
/*
Parsage
NB : Le serveur SOAP renvoie les valeurs lues
dans la table MySQL voitures dans un flux XML
avec comme balise d'encadrement <item> ... </item>
*/
var html = "";
// alert("Nombre de champs intégrés dans le flux XML : "
// + xmlDoc.getElementsByTagName("item").length);
for (i=0; i<xmlDoc.getElementsByTagName("item").length; i++)
{
    /* NB : childNodes[0] désigne le premier enfant
    de l'élément nommé item
    et nodeValue représente la valeur associée */
    if (i % 3 == 0)
    {
        html += "<br /><br />Code : "
        + xmlDoc.getElementsByTagName("item")[i]
        .childNodes[0].nodeValue;
    }
    if (i % 3 == 1)
    {
        html += "<br />Libellé : "
        + xmlDoc.getElementsByTagName("item")[i]
        .childNodes[0].nodeValue;
    }
    if (i % 3 == 2)
    {
        html += "<br />Vitesse maximale : "
        + xmlDoc.getElementsByTagName("item")[i]
        .childNodes[0].nodeValue;
    }
}
}
```

Le résultat du parsing est stocké dans une variable nommée `html`. Une boucle `for` balaie le flux et constitue progressivement le résultat.

Dans notre exemple, avec trois voitures caractérisées chacune par trois champs (`code_voiture`, `libelle_voiture` et `vitesse_maximale_voiture`) l'instruction :

```
xmlDoc.getElementsByTagName("item").length
```

donnera le résultat 9.

Le code :

```
xmlDoc.getElementsByTagName("item")[i].childNodes[0].nodeValue
```

représente la valeur de l'enfant direct de chaque item.

Un test (via une opération de modulo) permet de personnaliser chaque ligne intégrée dans la variable `html`. Concrètement lors du premier tour de la boucle `for` (`i` valant 0), la mention "Code :" sera prévue en préfixe.

Ce résultat (la variable `html`) sera ensuite reporté à l'écran dans une division HTML comme suit :

```
/* Placement du résultat du parsing dans la division HTML resultat */
var divisionResultat = document.getElementById("resultat");
divisionResultat.innerHTML = html;
```

Cette division d'affichage a été prévue dans la section HTML `<body> ... </body>` :

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

Étant donné les données intégrées dans le flux XML, l'affichage sera le suivant :

```
Code : V001
Libellé : Porsche 930 Turbo
Vitesse maximale : 290

Code : V002
Libellé : Porsche 964 Turbo
Vitesse maximale : 300

Code : V003
Libellé : Ferrari 430
Vitesse maximale : 320
```

Script PHP serveur_ws_01.php

Il nous reste à étudier le code du script côté serveur qui s'occupe de la récupération des données dans une table (voitures) hébergée sur une base de données MySQL et qui transmet un flux XML au script `client_ws_01.htm` dont nous venons de terminer le décryptage du code.

Bien évidemment, il ne peut s'agir au travers de ce script de vous former au développement applicatif par l'intermédiaire du langage de programmation PHP dans lequel il est écrit.

Un effort particulier a toutefois été réalisé, notamment dans la mise en place de commentaires, pour rendre ce script compréhensible par des débutants en PHP. Les points qui nécessiteraient une modification dans le cadre de la réutilisation dans un projet personnel seront mis en évidence.

Pour ce script, une reproduction du code complet est fournie ci-après. Des explications plus détaillées suivront.

```
<?php
/*
NOM DU SCRIPT : serveur_ws_01.php
REALISATION INFORMATIQUE : Christian VIGOUROUX
DATE DE CREATION : 01/01/2014
```

```

DATE DE DERNIERE MODIFICATION : 01/01/2014
OBJET : Listage de voitures de sport via un Webservice
(retour de champs en format XML)
*/

// Utilisation de la bibliothèque NuSOAP
require_once("nusoap.php");

// Création d'un objet SOAP (instanciation)
$serveur_soap = new soap_server;

// Enregistrement de la méthode listerVoitures dans l'objet
// afin qu'elle soit disponible pour les clients de cet objet
$serveur_soap->register("listerVoitures");

// Méthode listerVoitures
function listerVoitures()
{
    // 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;";
    // Paramètres SGBD MySQL
    $serveur_mysql = "sql.free.fr";
    $utilisateur_mysql = "christian.vigouroux";
    $mot_de_passe_mysql = "*****";
    $bdd_mysql = "christian_vigouroux";

    // Test de connexion à MySQL
    if (($connexion_mysql = @mysql_connect($serveur_mysql,
    $utilisateur_mysql, $mot_de_passe_mysql)) === FALSE)
    {
        // Message d'erreur envoyé au client
        return new soap_fault("Server", "MySQL", mysql_error());
    }
    else
    {
        // Test accès à la base de données
        if (@mysql_select_db($bdd_mysql, $connexion_mysql) === FALSE)
        {
            // Message d'erreur envoyé au client
            return new soap_fault("Server", "MySQL", mysql_error());
        }
        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
        $reponse[0] = "Table voitures vide";
        $reponse[1] = "0";
        return $reponse;
    }
else
{
    // Parcours séquentiel de l'extraction (vue SQL)
    $i = 0;
    $num_element = 0;
    while ($i < @mysql_num_rows($resultat_sql))
    {
        // Récupération des valeurs à afficher
        $code_voiture
        = mysql_result($resultat_sql, $i,
        "code_voiture");
        $libelle_voiture
        = mysql_result($resultat_sql, $i,
        "libelle_voiture");
        $vitesse_maximale_voiture
        = mysql_result($resultat_sql, $i,
        "vitesse_maximale_voiture");
        // Préparation envoi des résultats
        // au client
        $reponse[$num_element] = $code_voiture;
        $num_element = $num_element+1;
        $reponse[$num_element] = $libelle_voiture;
        $num_element = $num_element+1;
        $reponse[$num_element]
        = $vitesse_maximale_voiture;
        $num_element = $num_element+1;
        // Passage à l'enregistrement suivant
        $i++;
    }
    // Envoi de la réponse au client
    return $reponse;
}

}

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

}

// Envoi de la valeur de retour au client
$serveur_soap->service($HTTP_RAW_POST_DATA);

// Fin de code PHP
?>

```

Une première lecture du script vous a sans doute rassuré quant à la syntaxe propre à PHP. De nombreuses

similitudes existent avec JavaScript, notamment au niveau des structures de contrôles (`while { ... }, if (condition) { ... }, ...`).

Pour le nommage des variables mémoire, notez quelques particularités :

- les noms des variables doivent commencer par un \$,
- le langage est sensible à la casse (différenciation minuscules/majuscules),
- la déclaration (avant utilisation) des variables n'est pas requise.

Sachez aussi qu'en général le code PHP, comme le JavaScript d'ailleurs, vient s'insérer dans des séquences de code en HTML. Pour passer alternativement d'un code HTML à un code PHP dans le cadre d'un même script (ce qui n'est pas véritablement le cas dans notre exemple), il faut utiliser :

- `<?php` pour annoncer un début de séquence PHP,
- `?>` pour annoncer une fin de séquence PHP (passage à du HTML ou encore à du JavaScript).

L'extension du nom du script est normée (réglée par une directive du serveur Apache) et est en général `.php` ou encore `.php` suivi du numéro de la version de PHP utilisée (PHP 5 dans notre cas).

Enfin les règles de mise en commentaires sont communes à JavaScript (`/* ... */` ou `//`).

Passons en revue les principales instructions de ce script et en particulier les modifications que vous pourriez être amenés à effectuer dans le cadre d'une réutilisation personnelle.

La première séquence de code sert à intégrer dans le script PHP la bibliothèque de fonctions/méthodes `nusoap.php` (ce script est positionné dans le même répertoire que le script `serveur_ws_01.php`).

```
// Utilisation de la bibliothèque NuSOAP
require_once("nusoap.php");
```

Ensuite un objet de nom `$serveur_soap` est instancié à partir de la classe NuSOAP `soap_server`.

```
// Création d'un objet SOAP (instanciation)
$serveur_soap = new soap_server;
```

La séquence suivante déclare une méthode de nom `listerVoitures`. Cette méthode nous est déjà connue, nous l'avons vue dans la construction du message SOAP dans le script `client_ws_01.htm`.

```
// Enregistrement de la méthode listerVoitures dans l'objet
// afin qu'elle soit disponible pour les clients SOAP de cet objet
$serveur_soap->register("listerVoitures");
```



Dans le cadre d'une réutilisation du code de ce script serveur, le nom de cette méthode pourrait être différent.

Passons ensuite à la description de la méthode `listerVoitures`.

Vous retrouvez une syntaxe identique à celle de JavaScript pour décrire les fonctions, c'est-à-dire :

```
function nomFonction(paramètres) {  
    contenu de la fonction  
}
```

ou encore :

```
function nomFonction(paramètres)  
{  
    contenu de la fonction  
}
```

Dans notre cas particulier la fonction `listVoitures` ne reçoit pas de paramètre(s) du script client (`client_ws_01.htm`).

Une requête SQL est ensuite stockée dans une variable mémoire de nom `$requete_sql`. Cette requête sera tout naturellement soumise au moteur SQL du SGBD MySQL ultérieurement.

```
// 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;"
```

Dans le paragraphe suivant, les paramètres de connexion au serveur MySQL sont mémorisés dans des variables mémoire. Vous devrez en cas de mise en œuvre personnelle de ce script renseigner ces variables par les valeurs fournies par votre hébergeur MySQL. Dans la reproduction du script, le paramètre `$mot_de_passe` a été masqué.

```
// Paramètres SGBD MySQL  
$serveur_mysql = "sql.free.fr";  
$utilisateur_mysql = "christian.vigouroux";  
$mot_de_passe_mysql = "*****";  
$bdd_mysql = "christian_vigouroux";
```

La séquence suivante concerne la connexion au système de gestion de bases de données MySQL (il se nomme `sql.free.fr` dans notre cas, c'est le nom du calculateur MySQL mis à la disposition par le prestataire Free pour les hébergements gratuits). Bien évidemment les paramètres précédents (`$serveur_mysql`, `$utilisateur_mysql` et `$mot_de_passe_mysql`) vont être utilisés :

```
$connexion_mysql = @mysql_connect($serveur_mysql, $utilisateur_mysql, $mot_de_passe_mysql)
```

Si la connexion n'aboutit pas, une réponse est renvoyée vers le script client :

```
return new soap_fault("Server", "MySQL", mysql_error());
```

Dans le cas contraire, il reste à se connecter à la base de données (elle se nomme `christian_vigouroux` dans notre cas) dans laquelle la table `voitures` est entreposée. Rappelons que la connexion précédente se rapporte au SGBD uniquement, reste donc à assurer la connexion à la base comme suit :

```
@mysql_select_db($bdd_mysql, $connexion_mysql)
```

Comme pour la connexion au SGBD, la connexion à la base est testée et un message d'erreur est éventuellement renvoyé au script client :

```
return new soap_fault("Server", "MySQL", mysql_error());
```

Vous aurez remarqué que le retour d'erreur est programmé comme le précédent. Le serveur se charge de répercuter la nature de l'erreur via `mysql_error()`.

La séquence suivante est la plus intéressante, elle concerne le cas d'une connexion à la fois réussie au SGBD et à la base de données.

Dans un premier temps, la requête SQL est soumise au moteur SQL de MySQL :

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

Le résultat, qui est une structure matricielle entreposant les enregistrements en ligne et les champs en colonne, est stocké dans un tableau de nom `$resultat_sql`.

Si aucun enregistrement n'est trouvé (`@mysql_num_rows($resultat_sql)<1`) alors il faut le signaler à l'application cliente. Ceci est fait comme suit :

```
// Message d'erreur envoyé au client si pas d'enregistrement
$reponse[0] = "Table voitures vide";
$reponse[1] = "0";
return $reponse;
```

Il reste à traiter dans le dernier sinon (`else`) le cas le plus intéressant, celui de données trouvées (un ou plusieurs enregistrements).

Une structure itérative (`while`) est mise en place pour parcourir la structure matricielle `$resultat_sql`. La boucle s'exécute autant de fois qu'il y a d'enregistrements (`@mysql_num_rows($resultat_sql)`).

Pour chaque enregistrement il faut extraire la valeur pour chacun des champs (`code_voiture`, `libelle_voiture` et `vitesse_maximale_voiture` en ce qui nous concerne) comme suit par exemple pour le code de la voiture :

```
$code_voiture = mysql_result($resultat_sql, $i, "code_voiture");
```

Dans cette syntaxe `$code_voiture` est une variable temporaire pour mémoriser l'information, `mysql_result` la

fonction d'extraction, \$resultat_sql la matrice explorée, \$i le numéro de l'enregistrement traité (la numérotation débute à 0) et enfin "code_voiture" désigne le nom du champ concerné.

Il reste ensuite à placer ces différentes variables (\$code_voiture, \$libelle_voiture et \$vitesse_maximale_voiture) dans un tableau nommé \$reponse.

Ce tableau reponse est un tableau monodimensionnel qui comportera au final n cellules (trois champs * le nombre d'enregistrements).

L'alimentation du tableau \$reponse (cas du champ \$code_voiture) se fait comme suit :

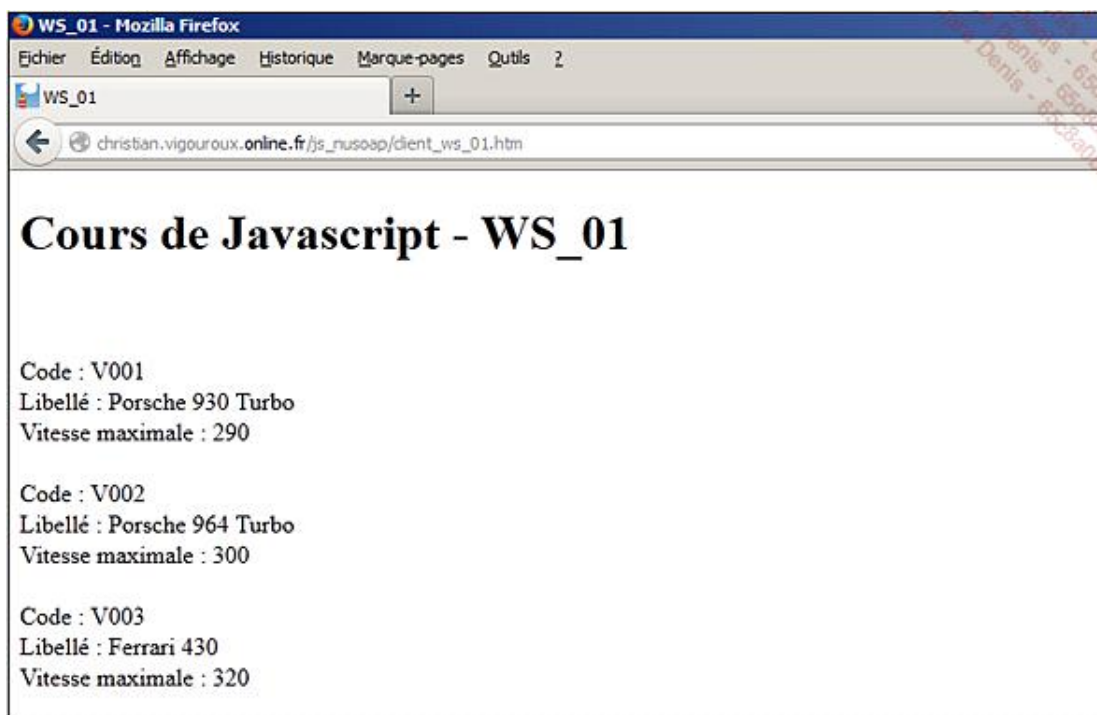
```
$reponse[$num_element] = $code_voiture;
```

À la fin du traitement, le tableau \$reponse sera renvoyé à l'application client :

```
return $reponse;
```

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_nusoap/, l'affichage obtenu à l'exécution du script client_ws_01.htm est le suivant :



2. Exemple 2 : Accès MySQL via Ajax

La principale différence dans ce deuxième exemple, par rapport au précédent, est que l'extraction de données depuis la table MySQL voitures ne porte plus sur l'intégralité des enregistrements mais sur un enregistrement unique pour lequel le code voiture aura été transmis depuis l'application cliente.

Les briques Ajax employées sont exactement les mêmes que dans l'exemple 1 (la réponse sera toujours renvoyée par le serveur sous la forme d'un flux XML).

L'exemple est donc l'occasion de voir comment passer un paramètre (le code de la voiture) à partir d'une application cliente à destination d'une application serveur.

Dans le commentaire des scripts constitutifs de cet exemple, nous nous focaliserons uniquement sur les aspects non vus dans le cadre du précédent exemple, donc principalement le passage de paramètre.

Script nusoap.php

Ce script PHP (bibliothèque de fonctions) a déjà été vu dans le cadre de l'exemple 1.

Script HTML client_ws_02.htm

Ce script client rédigé en HTML/JavaScript est structurellement très proche du script `client_ws_01.htm` vu dans l'exemple précédent.

Il intègre principalement deux fonctions JavaScript déjà vues dans l'exemple 1 :

- `appelerServeurDistant`
- `parserDOM`

Une nouvelle fonction, `contrôlerChoixListe`, vérifie la saisie effectuée du code de la voiture dont les données associées sont à rechercher dans la table MySQL voitures, ceci dans une liste déroulante et déclenchant finalement la recherche via un appel à la fonction `appelerServeurDistant`.

Dans la section HTML `<body> ... </body>`, un formulaire présentant une liste déroulante des codes des voitures et une division d'affichage du résultat (comme dans l'exemple 1) sont mises en place :

```
<!-- 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="contrôlerChoixListe(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 en HTML -->
  <div id="resultat"></div>
```

À chaque changement de code de voiture dans la liste déroulante nommée `liste`, la fonction `contrôlerChoixListe` est déclenchée. Le paramètre `getElementById('liste')` est passé à cette

fonction ainsi qu'un message alertant l'opérateur sur la nécessité de faire une sélection.

Notre application pourrait être améliorée en intégrant une liste déroulante des codes des voitures elle-même alimentée par une recherche sur le serveur distant. Pour ne pas trop alourdir l'exposé, ce choix technique n'a pas été effectué.

Parlons aussi des quelques variables communes aux fonctions JavaScript entreposées dans la section HTM <head> ... </head> :

```
/* Variables globales à toutes les fonctions JavaScript */
var httpRequest = null;
var valTimeout = null;
var url =
"http://christian.vigouroux.online.fr/js_nusoap/serveur_ws_02.php";
var messageSOAP = null;
```

Pas de surprise quant à ces variables, elles sont identiques à celles déjà vues dans l'exemple 1. La seule différence porte sur la valeur de la variable url, le script serveur qui va être sollicité est cette fois-ci serveur_ws_02.php.

Passons à l'étude détaillée des fonctions JavaScript assurant la récupération des données et leur traitement.

Fonction controlerChoixListe

Cette fonction n'était pas présente dans l'exemple 1, elle est donc reproduite intégralement ci-après :

```
/* 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
    {
        /* Affichage de contrôle */
        // alert("Le code voiture sélectionné est " +
        // document.getElementById('liste').value);
        // alert("Le code voiture sélectionné est " + liste.value);
        /* Appel du serveur distant */
        appelerServeurDistant(liste.value);
        /* Valeur de retour */
        return true;
    }
}
```

Il s'agit d'un code sans réelle surprise. Un test est effectué pour vérifier qu'un choix a effectivement été fait dans la

liste déroulante d'identifiant liste (la valeur par défaut étant associée à l'identifiant CODE). Si aucun code voiture n'a été sélectionné, le focus est replacé sur la liste après affichage d'un message d'alerte (messageAlerte). Dans le cas contraire, l'appel au serveur distant est effectué par la fonction appelerServeurDistant avec passage du code de la voiture sélectionnée (liste.value).

Fonction appelerServeurDistant

Le code est quasiment identique à celui étudié dans le cadre du premier exemple. La seule différence porte sur la construction du message SOAP à transmettre au serveur distant. Cette fois-ci il doit intégrer un paramètre, le code de la voiture dont les données sont à rechercher dans la table MySQL voitures.

```
/* Message SOAP à transmettre au serveur */
messageSOAP = '<?xml version="1.0" encoding="ISO-8859-1"?>';
messageSOAP += '<SOAP-ENV:Envelope SOAP-ENV:encodingStyle='
"http://schemas.xmlsoap.org/soap/encoding/"';
messageSOAP += ' xmlns:SOAP-ENV='
"http://schemas.xmlsoap.org/soap/envelope/"';
messageSOAP += ' xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"';
messageSOAP += ' xmlns:SOAP-ENC='
"http://schemas.xmlsoap.org/soap/encoding/"
xmlns:si="http://soapinterop.org/xsd">';
messageSOAP += '<SOAP-ENV:Body>';
messageSOAP += '<ns1:rechercherVoiture
xmlns:ns1="http://christian.vigouroux.online.fr">';
messageSOAP += '<type xsi:type="xsd:string">' + code_voiture + '</type>';
messageSOAP += '</ns1:rechercherVoiture>';
messageSOAP += '</SOAP-ENV:Body>';
messageSOAP += '</SOAP-ENV:Envelope>';
```

Finalement les seules lignes qui diffèrent sont celles-ci :

```
messageSOAP += '<ns1:rechercherVoiture
xmlns:ns1="http://christian.vigouroux.online.fr">';
messageSOAP += '<type xsi:type="xsd:string">' + code_voiture + '</type>';
messageSOAP += '</ns1:rechercherVoiture>';
```

La méthode du script serveur qui va être sollicitée s'appelle cette fois-ci rechercherVoiture (listerVoitures dans le cadre de l'exemple 1).

Vous remarquerez aussi l'intégration dans le message SOAP du paramètre code_voiture :

```
messageSOAP += '<type xsi:type="xsd:string">' + code_voiture + '</type>';
```

Rappelez-vous aussi que code_voiture est le paramètre passé par la fonction controlerChoixListe à la fonction appelerServeurDistant :

```
/* Appel du serveur distant */
```



```
appelerServeurDistant(liste.value);
```

```
/* Fonction appelerServeurDistant */  
function appelerServeurDistant(code_voiture)
```

Fonctions parserDOM et timeout

En ce qui concerne ces fonctions, il n'y a aucun changement par rapport à ce qui a été vu dans le cadre de l'exemple 1.

Script PHP serveur_ws_02.php

Le script serveur écrit en PHP et utilisant la librairie des fonctions NuSOAP est assez peu différent de celui vu dans l'exemple 1.

Le commentaire va donc porter sur les quelques différences.

La méthode `rechercherVoiture` est déclarée avec un passage de paramètre bien évidemment :

```
// Méthode rechercherVoiture  
// Paramètres passés par le client :  
// - $code_voiture : Code de la voiture de sport  
// dont on souhaite lister les caractéristiques  
function rechercherVoiture($code_voiture)
```

Un test est ensuite effectué pour vérifier qu'un paramètre a bien été passé (un message d'erreur est renvoyé à l'application cliente dans ce cas) :

```
// Test du code de la voiture fourni par le client  
if (empty($code_voiture))  
{  
  
    // Message d'erreur envoyé au client  
    return new soap_fault("Client", "Erreur voiture",  
        "Code voiture non renseigné");  
  
}  
else  
{  
    // ...
```

En réalité, dans notre application, ce test était un peu inutile dans la mesure où le choix du code de la voiture a été effectué dans le script client au travers d'une liste déroulante.

La requête SQL est aussi un peu modifiée, elle intègre le paramètre `$code_voiture` dans la clause SQL `where` :

```
// 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_voiture'";
```

C'est pratiquement terminé. Il reste quelques petites différences au niveau de la préparation de la réponse à fournir à l'application client.

Dans le cas (peu probable dans notre exemple) où la voiture n'est pas trouvée nous avons :

```
// Message d'erreur envoyé au client si pas d'enregistrement
$reponse[0] = "Voiture inexistante";
$reponse[1] = "0";
return $reponse;
```

et dans le cas favorable :

```
// Envoi du résultat au client
$reponse[0]
= mysql_result($resultat_sql, 0,
"code_voiture");
$reponse[1]
= mysql_result($resultat_sql, 0,
"libelle_voiture");
$reponse[2]
= mysql_result($resultat_sql, 0,
"vitesse_maximale_voiture");
return $reponse;
```

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_nusoap/, l'affichage obtenu à l'exécution du script `client_ws_02.htm` est le suivant :

