

Quelques propriétés détaillées

1. Propriété readyState

La propriété `readyState` de l'objet `XMLHttpRequest` permet de connaître l'évolution de la requête. Cette évolution prend successivement cinq valeurs reprises dans le tableau suivant :

Valeur	Description
0 ou uninitialized	La requête n'est pas initialisée.
1 ou loading	Début du transfert des données.
2 ou loaded	Les données sont transférées.
3 ou interactive	Les données reçues sont partiellement accessibles.
4 ou complete	Les données sont complètement accessibles.

C'est la valeur 4 ou *complete* qui retient notre attention car, une fois obtenue, les données transférées peuvent alors être traitées.

L'accès à la propriété `readyState` s'effectue par l'événement `onreadystatechange`, auquel doit être associée une fonction. Dans le cas présent, cette fonction permettra de connaître la valeur prise par `readyState`.

Le code devient :

```
if(window.XMLHttpRequest){
var xhr = new XMLHttpRequest();
}
else if(window.ActiveXObject){
var xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
else {
alert("Votre navigateur n'est pas compatible avec AJAX...");
}
xhr.onreadystatechange = function() {
if(xhr.readyState == 4) {
// instructions
}
}
xhr.open("GET", test.txt, true);
xhr.send(null);
```

Il faut noter que le code ajouté pour connaître l'état de la requête doit prendre place dans le script avant l'instruction `send()`.

Exemple 1

Au clic sur un bouton, testons si le fichier `test.txt` a bien été réceptionné (valeur 4 de `readyState`). Le résultat est affiché dans une boîte d'alerte.

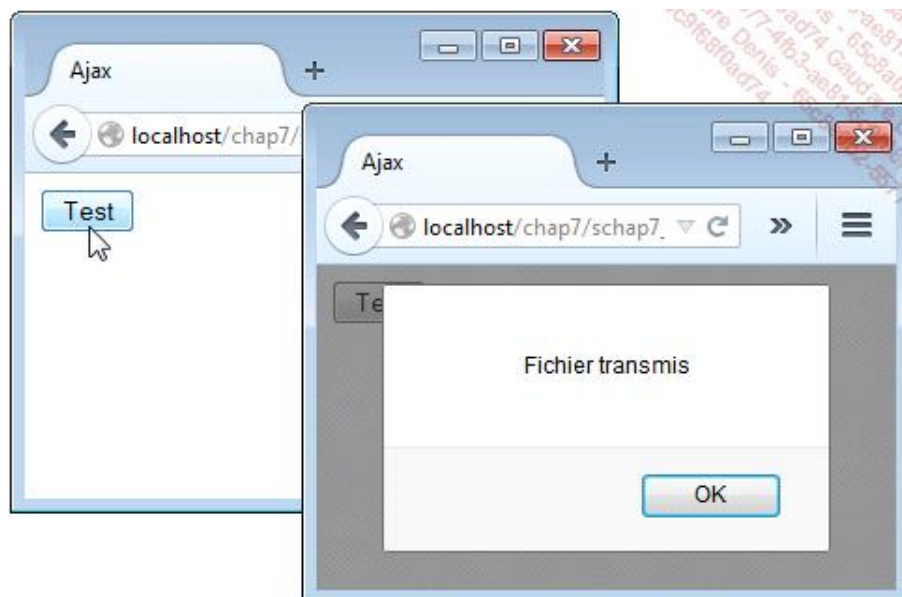
Le fichier `exemple.htm` :


```

<!DOCTYPE html>
<html lang="fr">
<head>
<title>XMLHttpRequest</title>
<meta charset="UTF-8">
<script>
function getxhr() {
var xhr=null;
if(window.XMLHttpRequest){
var xhr = new XMLHttpRequest();
}
else if(window.ActiveXObject){
var xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
else {
alert("Votre navigateur n'est pas compatible avec AJAX...");
}
xhr.onreadystatechange = function(){
if(xhr.readyState == 4) {
alert("Fichier transmis");
}
}
xhr.open("GET", "test.txt", true);
xhr.send(null);
}
</script>
</head>
<body>
<form>
<input type="button" value="Test" onclick="getxhr()">
</form>
</body>
</html>

```

Le fichier test.txt comporte les mots "Je viens du serveur".



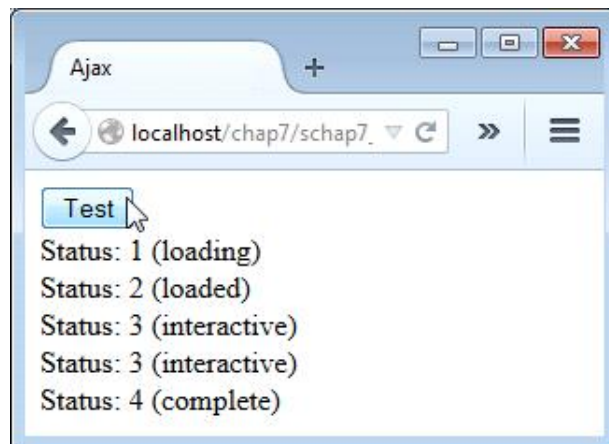
 Si vous modifiez le fichier HTM, il faut garder à l'esprit que le serveur local utilise la copie mise en cache du navigateur au lieu d'extraire de nouveau le fichier sur le disque dur, risquant ainsi de ne pas tenir compte des modifications apportées. Il sera ainsi parfois nécessaire, en cours de mise au point d'un script, de vider le cache du navigateur.

Exemple 2

Au clic sur un bouton, notons les différents états de la requête dans une balise <div> ... </div>. Le fichier test.txt est inchangé.

```
<!DOCTYPE html>
<html lang="fr">
<head>
<title>XMLHttpRequest</title>
<meta charset="UTF-8">
<script>
function getxhr(){
var xhr=null;
if(window.XMLHttpRequest){
var xhr = new XMLHttpRequest();
}
else if(window.ActiveXObject){
var xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
else {
alert("Votre navigateur n'est pas compatible avec AJAX...");
}
xhr.onreadystatechange = function(){
monDiv = document.getElementById("DivElement");
if(xhr.readyState == 1) {
monDiv.innerHTML += "Status: 1 (loading)<br>";
}
if (xhr.readyState == 2) {
monDiv.innerHTML += "Status: 2 (loaded)<br>";
}
if (xhr.readyState == 3) {
monDiv.innerHTML += "Status: 3 (interactive)<br>";
}
if (xhr.readyState == 4) {
monDiv.innerHTML += "Status: 4 (complete)<br>";
}
}
}
xhr.open("GET", "test123.txt", true);
xhr.send(null);
}
</script>
</head>
<body>
<form>
<input type="button" value="Test" onclick="getxhr()">
</form>
<div id="DivElement"></div>
```

```
</body>
</html>
```



Ne vous étonnez pas si vous n’obtenez pas exactement le même message. Selon le type et la version de votre navigateur et le degré d’intégration de l’objet XMLHttpRequest, certains états peuvent être ignorés.

2. Propriété status

La propriété `status` renvoie le code HTTP indiquant le résultat de la dernière requête exécutée.

Celui-ci vaut par exemple :

- 200 si la requête a été exécutée avec succès.
- 403 pour un accès interdit.
- 404 pour un fichier non trouvé.
- 500 pour une erreur interne au serveur.

Vous trouverez ci-dessous une liste plus détaillée des codes HTTP :

Code	Description
100	Continue
101	Switching protocols
200	OK
201	Created
202	Accepted
203	Non-Authoritative formation
204	No Content
205	Reset Content
206	Partial Content
300	Multiple Choices
301	Moved Permanently
302	Found

303	See Other
304	Not Modified
305	Use Proxy
307	Temporary Redirect
400	Bad Request
401	Unauthorized
402	Payment Required
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
407	Proxy Authentication Required
408	Request Timeout
409	Conflict
410	Gone
411	Length Required
412	Precondition Failed
413	Request Entity Too Large
414	Request-URI Too Long
415	Unsupported Media Type
416	Requested Range Not Suitable
417	Expectation Failed
500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Timeout
505	HTTP Version Not Supported

Une liste complète peut être consultée à l'adresse : http://fr.wikipedia.org/wiki/Liste_des_codes_HTTP

Les informations fournies par `readyState` sont, dans la pratique, trop partielles car la requête peut avoir été effectuée, mais sans succès (par exemple avec un code 404 fichier non trouvé). Ainsi, il est prudent de doubler le test `readyState == 4` par `status==200` qui confirme que la requête a bien été effectuée avec succès.

Le code devient alors :

```

if(window.XMLHttpRequest){
var xhr = new XMLHttpRequest();
}
else if(window.ActiveXObject){
var xhr = new ActiveXObject("Microsoft.XMLHTTP");
}

```

```

else {
alert("Votre navigateur n'est pas compatible avec AJAX...");
}
xhr.onreadystatechange = function() {
if(xhr.readyState == 4 && xhr.status == 200) {
// instructions
}
}
xhr.open("GET", test.txt, true);
xhr.send(null);

```

Le code complet de l'exemple précédent devient :

```

<!DOCTYPE html>
<html lang="fr">
<head>
<title>XMLHttpRequest</title>
<meta charset="UTF-8">
<script>
function getxhr(){
var xhr = null;
if(window.XMLHttpRequest){
var xhr = new XMLHttpRequest();
}
else if(window.ActiveXObject){
var xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
else {
alert("Votre navigateur n'est pas compatible avec AJAX...");
}
xhr.onreadystatechange = function(){
if(xhr.readyState == 4 && xhr.status == 200) {
alert("Fichier transmis");
}
}
xhr.open("GET", "test.txt", true);
xhr.send(null);
}
</script>
</head>
<body>
<form action="">
<input type="button" value="Test" onclick="getxhr()">
</form>
</body>
</html>

```

La capture d'écran est identique à celle du point précédent.

On peut aussi imaginer un autre code :

```

if (xhr.readyState == 4)

```

```

{
if (xhr.status == 200)
{
// instructions
}
}

```

Ici, le test pour vérifier que `status` vaut 200 ne s'effectue que lorsque le fichier a été transmis (`readystate == 4`).

3. Propriété `responseText`

La propriété `responseText` contient la réponse de la requête sous forme de chaîne de caractères (*String*).

Exemple 1

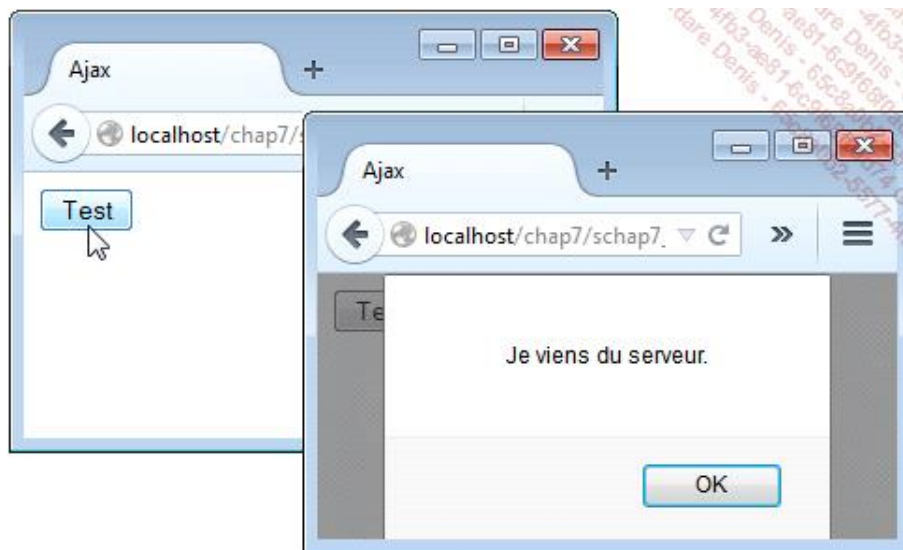
Au clic sur un bouton, affichons dans une boîte d'alerte le texte du fichier `test.txt`.

Le fichier `test.txt` comporte la phrase : "Je viens du serveur".

```

<!DOCTYPE html>
<html lang="fr">
<head>
<title>XMLHttpRequest</title>
<meta charset="UTF-8">
<script>
function getxhr(){
var xhr = null;
if(window.XMLHttpRequest){
var xhr = new XMLHttpRequest();
}
else if(window.ActiveXObject){
var xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
else {
alert("Votre navigateur n'est pas compatible avec AJAX...");
}
xhr.onreadystatechange = function(){
if(xhr.readyState == 4 && xhr.status == 200) {
alert(xhr.responseText);
}
}
xhr.open("GET", "test.txt", true);
xhr.send(null);
}
</script>
</head>
<body>
<form action="">
<input type="button" value="Test" onclick="getxhr()" />
</form>
</body>
</html>

```



Exemple 2

Au clic sur un bouton, affichons, dans une ligne de texte, le contenu du fichier test.txt. Le fichier test.txt comporte la phrase : "Je viens du serveur".

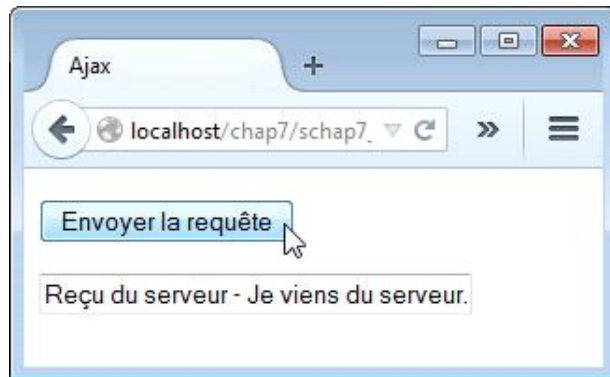
```
<!DOCTYPE html>
<html lang="fr">
<head>
<title>XMLHttpRequest</title>
<meta charset="UTF-8">
<script>
function getxhr(){
var xhr = null;
if (window.XMLHttpRequest){
xhr = new XMLHttpRequest();
}
else {
if (window.ActiveXObject){
xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
}
xhr.onreadystatechange = function(){
if(xhr.readyState == 4 && xhr.status == 200){
document.getElementById("txt").value = "Reçu du serveur - " +
xhr.responseText;
}
}
xhr.open("GET", "test.txt", true);
xhr.send(null);
}
</script>
</head>
<body>
<form name="ajax">
<p>
```



```


</p>
</form>
</body>
</html>

```



Exemple 3

Au clic sur un bouton, affichons, dans une boîte d’alerte, le texte du fichier test.xml.

Le fichier test.xml est le suivant :

```

<?xml version="1.0"?>
<compilation>
<mp3>
<titre>Foule sentimentale</titre>
<artiste>Alain Souchon</artiste>
</mp3>
<mp3>
<titre>Shape of my heart</titre>
<artiste>Sting</artiste>
</mp3>
<mp3>
<titre>Mistral gagnant</titre>
<artiste>Renaud</artiste>
</mp3>
</compilation>

```

Le code du fichier HTM est :

```

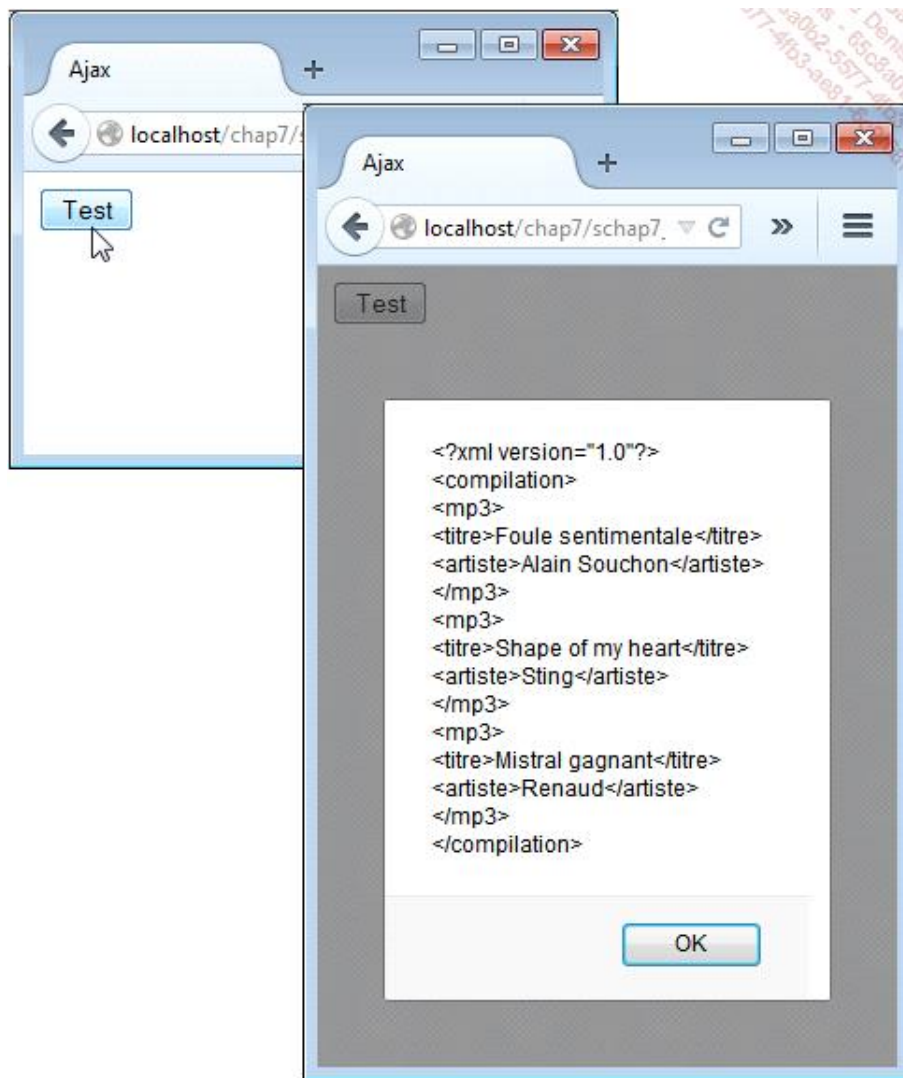
<!DOCTYPE html>
<html lang="fr">
<head>
<title>XMLHttpRequest</title>
<meta charset="UTF-8">
<script>
function getxhr(){

```

```

var xhr=null;
if(window.XMLHttpRequest){
var xhr = new XMLHttpRequest();
}
else if(window.ActiveXObject)
var xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
else {
alert("Votre navigateur n'est pas compatible avec AJAX...");
}
xhr.onreadystatechange = function(){
if(xhr.readyState == 4 && xhr.status == 200){
alert(xhr.responseText);
}
}
xhr.open("GET", "test.xml", true);
xhr.send(null);
}
</script>
</head>
<body>
<form>
<input type="button" value="Test" onclick="getxhr()">
</form>
</body>
</html>

```



Le fichier XML est retourné, dans cet exemple, comme une chaîne de caractères et non sous la forme d'un objet avec la structuration du fichier XML. Les éléments ne sont pas accessibles par des méthodes comme `getElementById`, `getElementsByTagName` ou équivalentes.

4. Propriété `responseXML`

La propriété `responseXML` renvoie les données comme un objet document XML. Il pourra alors être examiné et traité selon les méthodes et propriétés du DOM.

Soit avec le fichier XML suivant :

```
<?xml version="1.0"?>
<racine>
  Je suis un texte
</racine>
```

L'élément "Je suis un test" est accessible par le code :

```
var xmldoc = xhr.responseXML;
var root_node = xmldoc.getElementsByTagName('racine')[0];
alert(root_node.firstChild.nodeValue);
```

Nous reviendrons plus longuement sur l'accès aux données XML et leur traitement dans le chapitre suivant consacré à AJAX.



On peut rappeler lors de l'étude de ce chapitre que la méthode `XMLHttpRequest` porte finalement assez mal son nom car elle ne traite pas que du XML, mais aussi du texte et tous ses dérivés, comme des fichiers XHTML ou HTML, des fichiers JavaScript ou tout autre code.

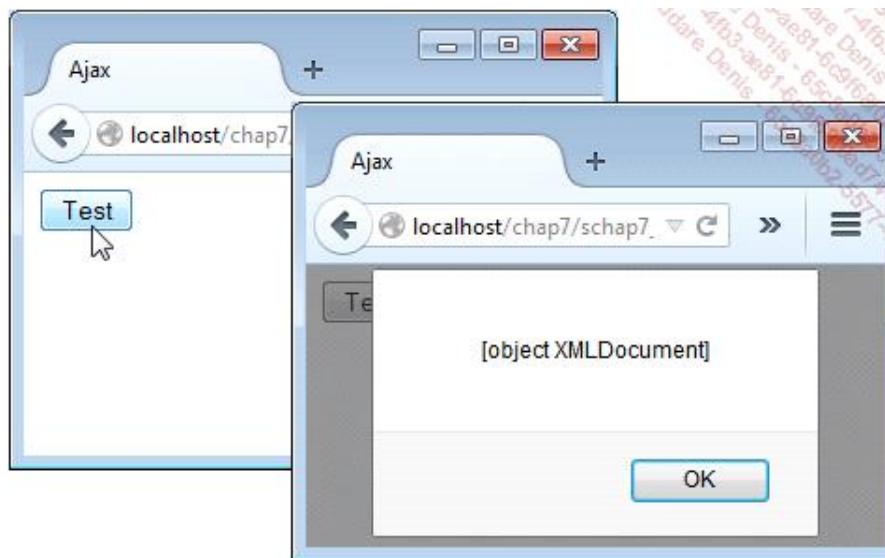
Exemple

Au clic sur un bouton, affichons, dans une boîte d'alerte, le texte du fichier `test.xml`.

Le fichier `test.xml` est identique à celui du point précédent.

Le code du fichier HTM est :

```
<!DOCTYPE html>
<html lang="fr">
<head>
<title>XMLHttpRequest</title>
<meta charset="UTF-8">
<script>
function getxhr(){
var xhr = null;
if(window.XMLHttpRequest){
var xhr = new XMLHttpRequest();
}
else if(window.ActiveXObject){
var xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
else {
alert("Votre navigateur n'est pas compatible avec AJAX...");
}
xhr.onreadystatechange = function(){
if(xhr.readyState == 4 && xhr.status == 200) {
alert(xhr.responseXML);
}
}
xhr.open("GET", "test.xml", true);
xhr.send(null);
}
</script>
</head>
<body>
<form>
<input type="button" value="Test" onclick="getxhr()">
</form>
</body>
</html>
```



Avec `responseXML`, le fichier XML est bien retourné comme un objet.

5. Propriété `timeout`

Parfois, lorsqu'une requête est initiée, la réponse du serveur peut être lente en fonction du taux d'occupation de celui-ci, ou du trafic sur la toile. L'utilisateur aura ainsi la désagréable impression que votre application bloque ou ne fonctionne pas. La version `XMLHttpRequest2` remédie à ce problème en introduisant la propriété `timeout`, qui permet de définir un temps maximum accordé à la réponse du serveur.

```
xhr.timeout = nombre
```

où `nombre` est le nombre de millisecondes que le navigateur peut attendre avant d'obtenir la réponse du serveur. La valeur par défaut est 0.

Ainsi, `xhr.timeout = 3000` détermine que le navigateur peut attendre 3 secondes. Passé ce délai, la requête sera arrêtée.

À cette propriété `timeout` est associé l'événement `ontimeout`, qui permettra d'associer une fonction au dépassement du délai fixé.

Le code pourrait être :

```
var xhr;
xhr = new XMLHttpRequest();
xhr.open("GET", url, true);
xhr.timeout = 10000;
xhr.ontimeout = function(evt){
    alert("Le délai de la requête est dépassé");
}
```

Un délai de 10 secondes est accordé. Passé ce délai, une fenêtre d'alerte s'affiche.