

# Primeros pasos en AJAX

*Ing. Juan Pablo Díaz Ezcurdia*  
[info@jpdiroz.com](mailto:info@jpdiroz.com)  
*Diciembre de 2009*

# Contenido

- I .-** Que es AJAX
- II.-** El objeto XMLHttpRequest
- III.-** Creación del objeto XMLHttpRequest
- IV.-** Realizar una petición con AJAX
- V.-** Recibir la petición AJAX
- VI.-** La respuesta AJAX
- VII.-** Tratamiento de la respuesta AJAX
- VIII.-** Mostrar los datos al usuario
- IX.-** Implementaciones de AJAX
- X.-** Prototype - Funciones Ajax

## I.- Que es AJAX

Según wikipedia AJAX (Asynchronous JavaScript And XML) es una técnica de desarrollo web para crear aplicaciones interactivas mediante la combinación de tres tecnologías ya existentes que conoceremos en este manual.

Es una manera de crear una aplicación que responde a las acciones del usuario sin refrescar la página contra el servidor.

### Tecnologías

Para conseguir este efecto, se utilizan la mayoría de las tecnologías disponibles para páginas web, HTML, CSS, XML, JavaScript y algún lenguaje de servidor cómo puede ser PHP o ASP, veamos qué función tiene cada lenguaje en la aplicación:

**JavaScript** - Para manejar el objeto XMLHttpRequest y DOM tratar para los datos recibidos.

**HTML** - Distribuye en la ventana del navegador los elementos de la aplicación y la información recibida por el servidor

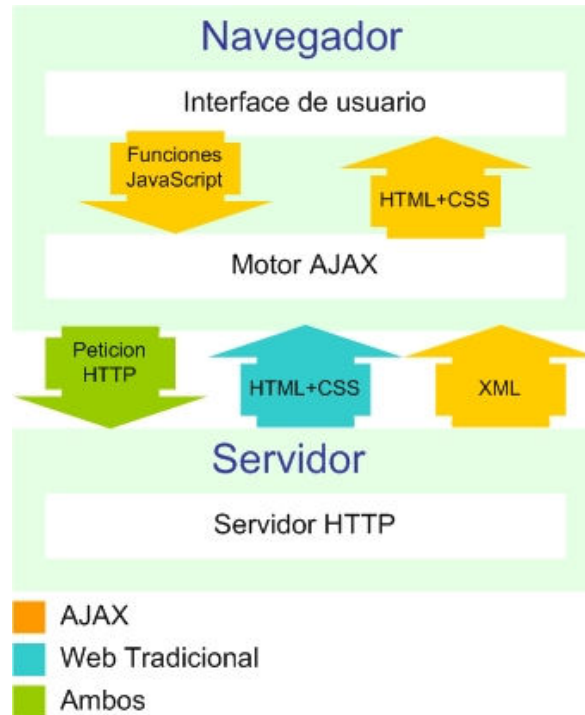
**CSS** - Define el aspecto de cada elemento y dato de la aplicación

**XML** - Es el formato de los datos transmitidos del servidor al cliente (navegador) y que posteriormente serán mostrados.

**Lenguaje de servidor** - Genera la información útil en XML y la envía al navegador.

### Funcionamiento

El usuario accede a la aplicación que es enviada por el servidor en formato HTML, JavaScript y CSS. Luego el código JavaScript de la aplicación pide al servidor los datos que quiere mostrar y este, ejecuta un código de lado de servidor que envía al navegador los datos en formato XML.



Cada vez que el usuario realiza una acción que significa mostrar unos datos, la capa javascript, repite la acción anterior de manera invisible al usuario y muestra los datos deseados.

## Problemas

El principal problema de la gran mayoría de aplicaciones AJAX (lo digo por experiencia como usuario de mozilla) es la baja compatibilidad entre navegadores, puesto que la capa JavaScript, es de una gran complejidad y a menudo por falta de experiencia en el lenguaje, o por falta de tiempo, se opta por programar solo para Internet Explorer.

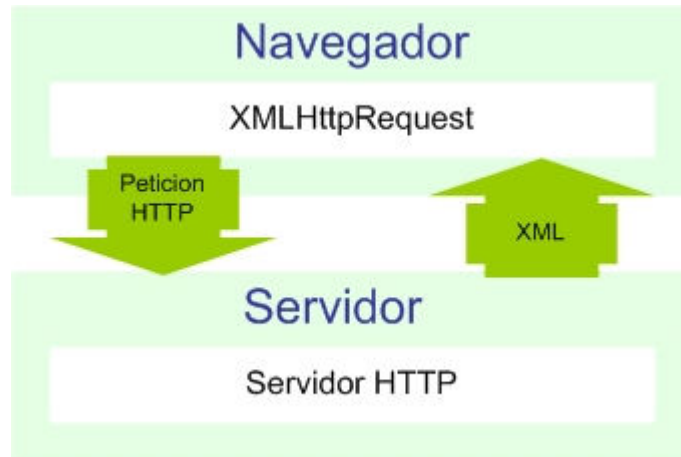
En futuras entregas, veremos cómo programar una aplicación AJAX compatible para todos los navegadores incluidos navegadores sin javascript.

## Ejemplos

Un excelente ejemplo de aplicación AJAX, bastante compatible entre navegadores es [Google Maps](http://Google Maps), en ella podrás ver cómo cambiamos la posición del mapa sin recargar la página.

## II.- El objeto XMLHttpRequest

Un objeto XMLHttpRequest es una instancia de una API que nos permite la transferencia de datos en formato XML desde los script del navegador (JavaScript, JScript, VBScript...) a los del servidor (PHP, Perl, ASP, Java...) e inversamente.



### Compatibilidad con navegadores

El primero en implementar esta API fue Microsoft con un objeto ActiveX para su navegador Internet Explorer 5, posteriormente empezó a incorporarse de forma nativa en todos los navegadores empezando por Firefox seguido de Apple, Konqueror, Opera Software, iCab y Microsoft Internet Explorer 7.

### Métodos y atributos

#### Atributos:

- onreadystatechange
- readyState
- responseText
- responseXML
- status
- statusText

#### Metodos:

- abort
- getAllResponseHeaders
- getResponseHeader
- open
- send
- setRequestHeader

### III.- Creación del objeto XMLHttpRequest

El secreto de AJAX es la comunicación sin refresco entre el cliente y el servidor, esto es posible gracias a JavaScript y al objeto XMLHttpRequest.

Este objeto, está disponible para la mayoría de navegadores modernos excepto las versiones 5 y 6 de Internet Explorer, para las cuales tendremos que usar un objeto ActiveX llamado 'Microsoft.XMLHTTP', por lo tanto, cuando creamos el objeto de comunicación con el servidor deberemos tener en cuenta el navegador con el que trabaja nuestro usuario.

Además, teniendo en cuenta que es posible que algunos usuarios accedan con un navegador sin JavaScript o con una versión pobre del mismo, en caso de que el objeto no pueda crearse de ninguna de las dos maneras, deberemos indicarlo al usuario o mejor todavía, dirigirlo a una versión tradicional de la aplicación (sin AJAX).

Para hacer el código más limpio, crearemos una función para realizar la conexión que usará variables locales, además es recomendable incluir todas las funciones que usaremos en un fichero .js externo e incluirlo en el documento HTML.

```
000 <script>
001 function AJAXCrearObjeto() {
002     var obj;
003     if(window.XMLHttpRequest) { // no es IE
004         obj = new XMLHttpRequest();
005     } else { // Es IE o no tiene el objeto
006         try {
007             obj = new ActiveXObject("Microsoft.XMLHTTP");
008         }
009         catch (e) {
010             alert('El navegador utilizado no está soportado');
011         }
012     }
013     return obj;
014 }
015 </script>
```

Ahora, llamaremos a la función AJAXCrearObjeto de la siguiente manera para obtener el objeto que utilizaremos más adelante:

```
000 <script>
001 oXML = AJAXCrearObjeto();
002 </script>
```

## IV.- Realizar una petición con AJAX

El primer paso para establecer la comunicación con el servidor usando AJAX, es hacer la petición, posteriormente, el servidor nos preparará y devolverá una información que ya veremos más adelante como recibimos, tratamos e incorporamos en nuestra página.

Existen dos tipos de petición al servidor que explicaremos en la referencia del método **open**, la petición síncrona y la asíncrona, pero por definición AJAX utiliza comunicación asíncrona que es la que explicaremos en este artículo.

### Realizar la petición al servidor

Para realizar la petición al servidor, utilizaremos los métodos **open**, **onreadystatechange** y **send**, que sirven respectivamente para preparar la petición, seleccionar la función de recepción e iniciar la petición.

Al método open, hay que pasarle el método de petición (GET) y la URL que se enviará al servidor y mediante la cual, el servidor, creará la respuesta que posteriormente leeremos.

Para nuestro primer ejemplo vamos a pedir un documento de texto:

```
000 <script>
001 // Creamos el objeto
002 oXML =AJAXCrearObjeto();
003 // Preparamos la petición
004 oXML.open('GET', 'archivo.txt');
005 // Preparamos la recepción
006 oXML.onreadystatechange = leerDatos;
007 // Realizamos la petición
008 oXML.send('');
009 </script>
```

Para que esto funcione, tendremos que haber declarado la función **leerDatos** para tratar los datos recibidos del servidor y mostrarlos al usuario, pero esto lo veremos más adelante.

### Paso de parámetros

En la petición AJAX podemos pasar parámetros tanto por POST como por GET a nuestro servidor.

Para [pasar parámetros por GET](#) ( por URL ) , usaremos una URL con parametros en la función **open** independientemente de usar el método GET o POST, por ejemplo:

```
000 <script>
001 // Creamos la variable parametro
```

```
002 parametro = 'Datos pasados por GET';
003 // Creamos el objeto
004 oXML = AJAXCrearObjeto();
005 // Preparamos la petición con parametros
006 oXML.open('GET', 'pagina.php?parametro=' + escape(parametro));
007 // Preparamos la recepción
008 oXML.onreadystatechange = leerDatos;
009 // Realizamos la petición
010 oXML.send('');
011 </script>
```

Para pasarlos por POST, deberemos usar el método POST en la función **open** y pasar los parámetros desde la función **send**, veamos un ejemplo:

```
000 <script>
001 // Creamos la variable parametro
002 parametro = 'Datos pasados por POST';
003 // Creamos el objeto
004 oXML = AJAXCrearObjeto();
005 // Preparamos la petición con parametros
006 oXML.open('POST', 'pagina.php');
007 // Preparamos la recepción
008 oXML.onreadystatechange = leerDatos;
009 // Realizamos la petición
010 oXML.send( 'parametro=' + escape(parametro));
011 </script>
```

**Nota:**

Siempre que enviemos parámetros, será conveniente preparar los datos previamente usando la función `escape`.

## V.- Recibir la petición AJAX

Vamos a ver como recibir la petición realizada en el anteriormente, recordamos que habíamos hecho una petición indicando que cuando cambie el estado de la misma, se ejecute la función *leerDatos*, a continuación vamos a ver cómo hacer esta función.

Lo primero que haremos será comprobar el estado de la petición y lo haremos con el método **readyState** que nos puede devolver cualquiera de los siguientes valores:

**0 (No inicializado)** - Los datos de la petición no se han definido

**1 (Abierto)** - La recepción de datos está en curso

**2 (Cargado)** - La recepción de datos ha finalizado pero los datos no están disponibles

**3 (Interactive)** - El objeto aún no está listo para otra petición pero ha recibido ya los datos.

**4 (Completado)** - El objeto está listo para otra petición

Y una vez estamos en estado cargado, ya podemos leer el texto recibido usando el método **responseText**, veamos un ejemplo:

```
000 function leerDatos(){
001     if (oXML.readyState == 4) {
002         alert (oXML.responseText);
003     }
004 }
```

Ahora vamos a ver el primer ejemplo completo de AJAX usando lo que hemos aprendido en este artículo y los dos anteriores de este mismo curso:

```
000 <script>
001 function leerDatos(){
002     if (oXML.readyState == 4) {
003         alert (oXML.responseText);
004     }
005 }
006 function AJAXCrearObjeto(){
007     var obj;
008     if(window.XMLHttpRequest) { // no es IE
009         obj = new XMLHttpRequest();
010     } else { // Es IE o no tiene el objeto
011         try {
012             obj = new ActiveXObject("Microsoft.XMLHTTP");
013         } catch (e) {
014             alert('El navegador utilizado no está soportado');
015         }
016     }
017     return obj;
018 }
019
020 oXML = AJAXCrearObjeto();
021 oXML.open('GET', 'archivo.txt');
022 oXML.onreadystatechange = leerDatos;
023 oXML.send('');
024 </script>
```

## VI.- La respuesta AJAX

Por definición, AJAX utiliza XML para organizar los datos transmitidos entre el servidor y el navegador, para que el navegador sea capaz de interpretar estos datos, tendrá que identificarlos cómo XML y su contenido tendrá que ser un XML válido, o de lo contrario, los datos no serán utilizables.

### Los encabezados

El primer paso para que el navegador interprete el contenido recibido es que tenga el encabezado de contenido XML (text/xml), esto lo conseguimos enviando desde el servidor el siguiente encabezado HTTP:

```
000 Content-Type: text/xml
```

Además, cómo nuestra respuesta XML será habitualmente generada de manera dinámica, es recomendable enviar también encabezamientos de control de caché para asegurarnos que la aplicación siempre estará trabajando con los contenidos que solicita y no con una cache almacenada en su navegador:

```
000 Cache-Control: no-cache, must-revalidate
001 Expires: Mon, 26 Jul 1997 05:00:00 GMT
```

Veamos cómo mandar estos encabezados con diferentes lenguajes de programación de lado de servidor, generalmente deberemos poner estos códigos al principio del todo del documento:

### PHP

```
000 <?php
001 header("Content-Type: text/xml");
002 header("Cache-Control: no-cache, must-revalidate");
003 header("Expires: Mon, 26 Jul 1997 05:00:00 GMT");
004 ?>
```

### Perl

```
000 #!/usr/bin/perl
001 print "Content-Type: text/xml";
002 print "Cache-Control: no-cache, must-revalidate";
003 print "Expires: Mon, 26 Jul 1997 05:00:00 GMT";
```

## ASP

```
000 <%
001 response.ContentType="text/xml"
002 response.CacheControl="no-cache, must-revalidate"
003 response.Expires="Mon, 26 Jul 1997 05:00:00 GMT"
004 %>
```

## JSP

```
000 <%
001 response.setHeader("Content-Type", "text/html;charset=windows-
002 1252");
003 response.setHeader("Expires", "Mon, 01 Jan 2001 00:00:01 GMT");
004 response.setHeader("Cache-Control", "must-revalidate");
005 response.setHeader("Cache-Control", "no-cache");
006 %>
```

## El contenido

Cuando el navegador recibe el contenido en XML, lo analizará para estructurar los datos recibidos para que podamos tratarlos desde nuestra aplicación, para que esto funcione, el contenido del documento deberá ser XML válido y por lo tanto, deberá empezar con la declaración de versión:

```
000 <?xml version="1.0"?>
```

### Nota:

Debemos tener cuidado con la declaración de XML cuando trabajamos con archivos PHP, porque PHP interpreta **<?** como inicio de su código cuando tiene las short tags activadas.

Ahora podremos enviar los datos en formato XML correcto (podemos utilizar el validador de XML del consorcio W3C), veamos un ejemplo:

```
000 <?xml version="1.0"?>
001 <xml>
002   <mensaje>
003     <color>#000000</color>
004     <texto>Texto del mensaje</texto>
005   </mensaje>
006 </xml>
```

## VII.- Tratamiento de la respuesta AJAX

Una vez recibida la petición AJAX debemos saber interpretar los datos XML recibidos usando JavaScript, para ello, utilizaremos **responseXML** en lugar de **responseText**, y podremos empezar a parsear el XML recibido:

```
000 <script>
001 function leerDatos(){
002     if (oXML.readyState == 4) {
003         var xml = oXML.responseXML.documentElement;
004         // ...
005     }
006 }
007 </script>
```

Los ejemplos de este artículo los haremos pensando en el siguiente xml:

```
archivo.xml
000 <?xml version="1.0" encoding="UTF-
001 8" standalone="yes"?>
002 <xml>
003     <mensaje>
004         <texto>Ejemplo 1</texto>
005     </mensaje>
006     <mensaje>
007         <texto>Ejemplo 2</texto>
008     </mensaje>
009 </xml>
```

### Acceso a un elemento XML

A partir de este momento, la variable *xml* (*responseXML.documentElement*) será una referencia al documento XML recibido, y nos permitirá el acceso a los datos enviados por el servidor en forma de documento XML usando DOM.

#### Nota:

Para poder tratar los datos recibidos es importante conocer DOM. Antes de continuar puedes consultar estos enlaces:

Veamos un ejemplo de acceso a un elemento mensaje:

```
000 <script>
001 var item = xml.getElementsByTagName('mensaje')[0];
002 </script>
```

La función **getElementsByTagName** nos devuelve un array con todos los elementos con el nombre de tag indicado, en este caso serán los elementos <mensaje>.

Podemos acceder a un elemento determinado poniendo un número entre llaves (en el ejemplo accedíamos al 0) o usar un bucle para recorrer todos los elementos:

```
000 <script>
001 for (i = 0; i < xml.getElementsByTagName('mensaje').length; i++){
002   var item = xml.getElementsByTagName('mensaje')[i];
003 }
004 </script>
```

Podremos también acceder a un subelemento de la misma manera:

```
000 <script>
001 var item = xml.getElementsByTagName('mensaje')[0];
002 var txt = item.getElementsByTagName('texto')[0];
003 </script>
```

### Acceso al texto de un elemento

Para acceder al texto entre las etiquetas <texto> y </texto> usaremos **firstChild.data** sobre el elemento:

```
000 <script>
001 var item = xml.getElementsByTagName('mensaje')[0];
002 var txt = item.getElementsByTagName('texto')[0];
003 alert(txt.firstChild.data)
004 </script>
```

### Ejemplo completo

Este es el archivo del ejemplo que utiliza lo explicado en este artículo para parsear el XML anterior:

```
index.html
000 <html>
001 <head>
002 <title>ProgramaciónWeb - Ejemplo</title>
003 <script>
004 function leerDatos(){
005   if (oXML.readyState == 4) {
006     var xml = oXML.responseXML.documentElement;
007     for (i = 0; i < xml.getElementsByTagName('mensaje').length; i++){
008       var item = xml.getElementsByTagName('mensaje')[i];
009       var txt = item.getElementsByTagName('texto')[0].firstChild.data;
010       alert(txt);
011     }
012   }
013 }
```

```
014 function AJAXCrearObjeto(){
015 var obj;
016 if(window.XMLHttpRequest) { // no es IE
017 obj = new XMLHttpRequest();
018 } else { // Es IE o no tiene el objeto
019 try {
020 obj = new ActiveXObject("Microsoft.XMLHTTP");
021 }
022 catch (e) {
023 alert('El navegador utilizado no está soportado');
024 }
025 }
026 return obj;
027 }
028
029 oXML = AJAXCrearObjeto();
030 oXML.open('get', 'archivo.xml');
031 oXML.onreadystatechange = leerDatos;
032 oXML.send('');
033 </script>
034 </head>
035 </html>
```

## VIII.- Mostrar los datos al usuario

Una vez hemos recibido y conocemos los datos que necesitamos del servidor, deberemos mostrarlos al usuario de alguna manera.

En la mayoría de casos, lo que nos interesa, no es mostrar estos datos en un mensaje emergente usando la función **alert** (cómo hemos visto en los ejemplos anteriores), sino que queremos mostrar los datos en la misma página que está viendo el usuario sin usar refresco.

### ¿Dónde mostramos los datos?

Para mostrar estos datos, necesitamos un objeto HTML al que le podamos modificar su contenido de manera que el usuario pueda verlo.

Normalmente aunque no siempre, usaremos un objeto **div** al que le podremos modificar su contenido usando el atributo **innerHTML**:

```
000 <div id="miDiv1">Aquí aparecerán los datos</div>
```

### ¿Cómo mostramos los datos?

Para mostrar los datos que hemos obtenido en el **div** que hemos creado, primero accederemos al objeto a través de su id (miDiv1 en el ejemplo) usando el método getElementById y luego podremos usar **innerHTML** para indicarle el contenido en formato HTML que tendrá este **div** en su interior:

```
000 <script>
001 // Accedemos al DIV con getElementById
002 miDiv = document.getElementById('miDiv1');
003 // Modificamos su contenido
004 miDiv.innerHTML = '<b>Este es el nuevo contenido</b>';
005 </script>
```

La manera como pasaremos los datos recibidos del servidor al div, dependerá de cada caso, pero vamos a ver un ejemplo que puede ser útil, imaginemos que tenemos esta lista de usuarios en XML:

```
usuarios.xml
000 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
001 <xml>
002   <usuario>
003     <id>1</id>
004     <nombre>Eloi</nombre>
005   </usuario>
006   <usuario>
007     <id>2</id>
008     <nombre>Pedro</nombre>
009   </usuario>
010   <usuario>
```

```
011 <id>3</id>
012 <nombre>Juan</nombre>
013 </usuario>
014 </xml>
```

Vamos a suponer que hemos pedido estos datos al servidor y que hemos indicado como `readystatechange` handler la función `leerDatos`, como vimos en el artículo [Recibir la petición AJAX](#) y vamos a centrarnos en el contenido de dicha función para que muestre una lista de usuarios con enlace a su perfil en el div que hemos creado con id `miDiv1`:

```
000 <script>
001 // Recibe y muestra los datos
002 function leerDatos(){
003     // Comprobamos que se han recibido los datos
004     if (oXML.readyState == 4) {
005         // Accedemos al XML recibido
006         var xml = oXML.responseXML.documentElement;
007         // Accedemos al DIV
008         var miDiv = document.getElementById('miDiv1');
009         // Vaciamos el DIV
010         miDiv.innerHTML = '';
011         // Iteramos cada usuario
012         for (i = 0; i < xml.getElementsByTagName('usuario').length; i++){
013             // Accedemos al objeto XML usuario
014             var item = xml.getElementsByTagName('usuario')[i];
015             // Recojemos el id
016             var id = item.getElementsByTagName('id')[0].firstChild.data;
017             // Recojemos el nombre
018             var nombre = item.getElementsByTagName('nombre')[0].firstChild.data;
019             // Mostramos el enlace
020             miDiv.innerHTML += '<a href="/perfil/'+id+'/">'+nombre+'</a><br>';
021         }
022     }
023 }
024 </script>
```

## IX.- Implementaciones de AJAX

Existen muchas implementaciones de AJAX muy interesantes que podemos encontrar por Internet y nos facilitarán el desarrollo de aplicaciones con comunicación con el servidor.

Vamos a hacer una lista de las que consideramos más interesantes, las ordenaremos en dos grupos librerías de cliente y librerías de cliente/servidor:

### Librerías de cliente

Estas nos permiten trabajar fácilmente con llamadas al servidor y tratar los datos recibidos, la ventaja de estas es que no dependen de un lenguaje de servidor pero por esta misma razón suelen integrarse peor.

**prototype** - Esta interesante librería de JavaScript dispone de ( entre otros ) la clase Ajax, que nos facilitarán muchísimo el trabajo en AJaX

### Librerías de cliente/servidor

Este tipo de librerías nos permite hacer llamadas a funciones del servidor desde el cliente usando unas funciones JavaScript autogeneradas a las que llamaremos proxies.

**Ajax.NET Professional** - Conocido como AjaxPro nos permite crear una proxy para llamar a métodos .Net del servidor desde JavaScript.

**xaJax** - xaJax es un framework (marco de trabajo) escrito en PHP de código abierto que permite crear fácilmente aplicaciones web que utilizan AJaX

**XOAD** - XOAD es un proxy de PHP usando XMLHttpRequest y JSON

En los próximos capítulos hablaremos sobre alguna de estas librerías, si no te interesa puedes pasar al cuestionario de AJaX

## X.- Prototype - Funciones Ajax

Las funciones **Ajax de Prototype**, simplifican la comunicación Ajax con el servidor y la inserción de los datos recibidos en el documento actual.

### Lista de opciones:

Los constructores de las clases prototype que describimos a continuación pueden recibir las siguientes opciones:

**asynchronous** ( true | false ) - Realizar la petición en modo síncrono o asíncrono, por defecto *true*

**contentType** ( string ) - Tipo mime de la petición, por defecto *'application/x-www-form-urlencoded'*

**encoding** ( string ) - Codificación de caracteres de la petición, por defecto *'UTF-8'*

**method** ( string ) - Método de la petición ( *'GET'* o *'POST'* )

**parameters** ( string o objeto ) - Parámetros como string tipo *'?num=1&page=0'* o tipo *{ 'num':1, 'page':0 }*

**postBody** ( string ) - Cuerpo de la petición en caso de usar **method** *'POST'* .

**requestHeaders** ( array u objeto ) - Parámetros HTTP adicionales de la petición como array u objeto tipo *{ 'Accept':'text/javascript' }*

**Eventos** ( función ) - Eventos sobre la petición: **onComplete**, **onException**, **onFailure**, **onInteractive**, **onLoaded**, **onLoading**, **onSuccess**, **onUninitialized** y **onNNN** ( donde NNN es un HTTP Status Code )

### Lista de clases:

**clase [font color="#008000"]Ajax.PeriodicalUpdater[/h1]Realiza cada N segundos una petición AJAX y rellena el elemento HTML indicado con la respuesta recibida.**

**Los parámetros del constructor son:**

**[b]1** - ID del elemento HTML a rellenar

**2** - URL de la petición

**3** - Opciones de la lista de opciones anterior y estas dos específicas:

**frequency** ( entero ) - Tiempo en segundos entre peticiones

**decay** ( entero ) - El valor de **frequency** se multiplica por este cada vez que se recibe una respuesta sin modificaciones respecto a la anterior.

```
000 new Ajax.PeriodicalUpdater('idMiElemento', '/elementos.php', {
001   method: 'get', frequency: 3, decay: 2
002 });
```

**clase [font color="#008000"]Ajax.Request[/h1]Realiza una petición AJAX.**

**Los parámetros del constructor son:**

**[b]1** - URL de la petición

**2** - Opciones de la lista de opciones anterior.

```
000 new Ajax.Request( '/elementos.php' , {
001   method: 'get',
002   onSuccess: function(transport) {
003     alert ( transport.responseText ) ;
004   }
005 });
```

**función [font color="#008000"]Ajax.Responders.register[/h1]Permite registrar eventos comunes que se lanzarán cuando se produzca un determinado evento para cualquier petición.**

**Los parámetros del constructor son:**

**[b]1** - URL de la petición

**2** - Opciones de la lista de opciones anterior.

```
000 canalesAjax = 0;
001 Ajax.Responders.register({
002   onCreate: function() {
003     canalesAjax++;
004     alert ( canalesAjax ) ;
005   },
006   onComplete: function() {
007     canalesAjax--;
008   }
009 });
```

**función [font color="#008000"]Ajax.Responders.unregister[/h1]Cancela un evento registrado con [b]Ajax.Responders.register**

**clase [font color="#008000"]Ajax.Updater[/h1]Realiza una petición AJAX y rellena el elemento HTML indicado con la respuesta recibida.**

**Los parámetros del constructor son:**

**[b]1** - ID del elemento HTML a rellenar

**2** - URL de la petición

**3** - Opciones de la lista de opciones anterior y estas dos específicas:

**evalScripts** ( true | false ) - Ejecutar los <script> recibidos en la petición.

**insertion** ( objeto Insertion ) - Si se especifica, en lugar de reemplazar el contenido de **1** inserta los datos en la posición indicada por el objeto.

```
000 new Ajax.Updater('idMiElemento', '/elementos.php', {
001   method: 'get',
002   insertion: Insertion.Bottom
003 });
```