

PHP - MySQL - JavaScript (XAJAX)

XAJAX

José Luis Comesaña



13

INDICE

Qué permite hacer? _____	3	browserSupportsCookies() _____	40
Instalación del zip _____	3	browserSupportsAjax() _____	40
Funcionamiento _____	3	ActiveXEnabledOrUnnecessary() _____	41
Funciones PHP _____	4	Interfaz de navegación por pestañas _____	41
Métodos de Creación _____	5	Maquetación de la interfaz con CSS y HTML	41
Métodos de Eliminación _____	6	Código PHP de la función para cambiar la	
Métodos de Modificación _____	6	pestaña pulsada _____	42
Métodos de Interacción con JavaScript _____	6	Código completo de la interfaz de pestañas	43
Métodos de Adición de Eventos _____	7	Sistema de votación con Xajax _____	45
XAJAX - EL OBJETO _____	7	Creación del recuadro de votación _____	45
XAJAX - EN EL LADO CLIENTE _____	9	Explicación del código _____	47
Ejemplo _____	10	Sistema de votación con Xajax mejorado	49
Página simple con xajax y PHP _____	13	Incluir un archivo de texto o HTML a través	
Recibir formulario con Ajax e insertar		de una llamada a xajax _____	54
información en base de datos. _____	15	Mensaje de carga con ajax usando xajax	55
Enviar los datos del formulario a PHP con		Código para mostrar un mensaje de carga	
xajax _____	15	con Xajax _____	57
Recibir los datos del formulario enviado con		Sistema de ordenación de elementos _____	58
xajax _____	16	Actualizar Xajax 0.2 a Xajax 0.5 _____	62
Envío de formulario e insertar la		Instalar Xajax 0.5 _____	62
información en una base de datos MySQL.	17	Actualizar los includes _____	62
Procesar formulario Ajax con validación	19	3. Actualiza la sintaxis _____	62
El formulario y la zona de mensajes _____	19	Métodos de la clase xajax _____	62
Solución para los acentos en xajax _____	20	Métodos de la clase xajaxResponse _____	63
Select combinado _____	22	Ejemplo de actualización de script Xajax _____	63
Comprobar si un usuario es válido _____	25	Aviso de cargando para Ajax con Xajax 0.5	64
Mostrar un formulario al pulsar un enlace	28	Procesar formularios múltiples con Xajax	
Manejo de sesiones _____	30	0.5 _____	67
Función para iniciar el juego. _____	31	Construcción de los formularios _____	67
Función para probar una letra y ver si está en		Función Xajax para procesar el formulario _____	68
la palabra buscada. _____	32	Tratamiento UTF-8 en Xajax 0.5 _____	69
Conclusión sobre variables de sesión en Ajax	32	Todo el código junto _____	69
Comprobar la validez de una URL _____	35	Cambiar el estilo CSS de una página _____	70
Validación previa de un campo de texto _____	37	Generar eventos a elementos _____	72
El formulario HTML _____	37	Métodos addEvent() y addHandler() de	
La llamada Ajax para validar el país _____	37	xajaxResponse _____	72
Función para procesar el formulario con Ajax		Otros detalles a comentar sobre la	
y PHP _____	38	declaración de eventos con Xajax _____	74
Javascript para escoger el país deseado _____	38	Anexo I - Aprenda XAJAX en 10 minutos	75
Script para detección de soporte a Ajax,		Usando xajax en un script PHP _____	75
Cookies y ActiveX _____	40	¿Cómo actualizo mi contenido	
		asíncronamente? _____	75

XAJAX

xaJax es un framework (marco de trabajo) escrito en PHP de código abierto que permite crear fácilmente aplicaciones web que utilizan AJAX sin necesidad siquiera de conocer JavaScript (aunque sí que hay la posibilidad de hacer funciones en este lenguaje). Puedes bajarte el zip desde <http://www.xajaxproject.org>.

Qué permite hacer?

En una aplicación AJAX, el servidor crea una página que es enviada al cliente. Éste interactúa con la página (rellena formularios, hace clic en ciertos objetos) que disparan ciertos eventos (onclick, onchange...) que llaman a funciones JavaScript. Estas funciones pueden o no interactuar con el servidor (usando AJAX) y recibiendo información de éste, mostrándola al usuario cambiando el contenido de la misma página.

Este tipo de páginas resultan muy útiles y son uno de los pilares de la Web 2.0. Aún así, realizar un proyecto AJAX puede resultar muy costoso y largo; pero no siempre es así.

XaJax te permite escribir funciones en PHP que pueden ser accedidas cuando ya la página ha estado enviada al navegador, cuando el usuario ha disparado un evento o la función PHP ha sido llamada desde JavaScript. Éstas funciones PHP modifican el contenido o el aspecto de la página, como lo podría hacer JavaScript, pero repito que se hacen desde PHP, lo que abre un abanico de posibilidades: PHP puede acceder a bases de datos, al sistema de archivos... Y todo sin recargar la página!!!

Instalación del zip

Para instalar la librería xaJax, debes abrir el zip que te descargas de la web de xaJax y descomprimirlo conservando la estructura de las carpetas. Descomprímelo desde la carpeta donde tienes la web en servidor local o remoto y cámbiale el nombre de la carpeta (normalmente el nombre xajax seguido del nombre de la versión) por solamente xajax.

La versión 0.2 del zip consta de tres scripts PHP: `xajax.inc.php`, que contiene la clase `xajax`, `xajaxResponse.inc.php`, que contiene la clase `xajaxResponse`, y `xajaxCompress.php`, que es una utilidad para, de cierta manera, reducir el peso de un archivo JavaScript. Contiene también, en la carpeta `xajax_js`, dos archivos `.js`; uno que contiene el código original (el `xajax_uncompressed.js`) y otro que es el que está comprimido y es el que se usa para enviar al navegador (el `xajax.js`). Además de eso, también incorpora dos archivos `.txt` (la licencia y el `readme`) y dos carpetas más (`examples`, con ejemplos de utilización y `tests`, donde puedes probar xaJax).

Funcionamiento

Lo único que hay que hacer es hacer un `require_once()` al `xajax.inc.php` de la carpeta donde lo has instalado en el servidor, local o remoto, y jugar con las posibilidades que te brindan los dos objetos: `xajax` y `xajaxResponse` (éste integrado dentro de la función PHP).

Como he dicho antes, xaJax permitirá a una función JavaScript recibir una respuesta de una función del servidor. Dicha respuesta estará en formato XML (como es habitual en AJAX) y será

interpretada por la misma función JavaScript que realizará los cambios en la página que se le piden.

Así, la respuesta (en formato XML) dará unas instrucciones específicas tales como crea una etiqueta `div` aquí, dale este formato y ponle este texto, elimina la etiqueta con el `id` tal, o haz que se ejecute esta función JavaScript.

De cierta manera, el objeto `xajaxResponse` contiene la respuesta que realiza la función PHP y el objeto `xajax` es el que recibe las peticiones de que se ejecuten las funciones, el que las gestiona, al igual que todas las respuestas, y el que envía las respuestas en XML de vuelta al navegador.

Por lo tanto, podemos distinguir dos partes en una página basada en xAJax: una parte sería todo el código php que contiene las funciones y los objetos `xajax` y `xajaxResponse` (además de todo el código necesario para que la web tenga un entorno dinámico) y otra parte que sería todo lo que se ejecuta en el navegador.

Traducción de un tutorial de la página oficial

<http://jvelazqu.glo.org.mx>

Funciones PHP

Antes de leerte éste artículo, sería interesante que te leyeras el anexo:

Aprenda XAJAX en 10 minutos

Una vez leído, puedes leerte éste que amplía en gran medida el del anexo.

Las funciones xAJax son funciones escritas en PHP que son llamadas desde JavaScript. Estas funciones tienen dos objetivos: interactuar con el servidor (por eso se ha llamado a ésta función, para crear un usuario nuevo, para examinar los archivos que hay en un directorio...) y devolver una respuesta XML que será enviada al navegador y interpretada por JavaScript.

Esta respuesta será hecha por el objeto `xajaxResponse`, el que se configurará con métodos y creará el XML que será enviado al navegador.

El constructor del `xajaxResponse` consta de dos argumentos:

- ✓ La codificación (*por defecto "utf-8"*)
- ✓ Un booleano que si es `true` muestra los caracteres especiales en el navegador y si es `false` los oculta (*por defecto, los oculta*).

El `xajaxResponse` será el encargado de, una vez configurado por nosotros a partir de métodos, crear una respuesta XML. Éste XML será el que tendrá de ser devuelto al navegador, e interpretado por el JavaScript. Así que ya podemos hacer el esquema de una función xAJax:

```
function esquema(arg1, arg2...){  
    // Aquí iría el código destinado a realizar el primer objetivo  
    // de una función xAJax, el de interactuar con el servidor  
    // actuando con los argumentos de la función o lo que haga falta.  
  
    // Se crea un nuevo objeto xajaxResponse al que se le podrían  
    // pasar como atributos la codificación y el valor booleano que  
    // comentábamos antes.  
    $respuesta = new xajaxResponse();  
  
    // Aquí iría el código para modificar y configurar el objeto, y
```

```
// al mismo tiempo el XML

// Y finalmente retornamos la respuesta XML.
return $respuesta->getXML();
```

Desde la versión 0.2, no hace falta llamar al método `getXML()` para retornar, tan solo hace falta retornar el objeto `xajaxResponse` poniendo en este caso `return $respuesta;`

El código para modificar el `xajaxResponse` y en consecuencia la respuesta XML, está todo basado en métodos del mismo objeto. Todos añaden un comando al XML que será interpretado y ejecutado en el navegador por JavaScript.

Estos comandos (escritos por métodos) permiten cantidad de cosas, todas orientadas a cambiar el aspecto de la página.

Métodos de Creación

Los métodos de creación permiten crear una etiqueta dentro del árbol de etiquetas HTML. Ésto se puede conseguir con el método `addCreate($id_etiqueta_superior, $etiqueta, $id)`, que creará una etiqueta del tipo `$etiqueta` dentro de la etiqueta especificada por `$id_etiqueta_superior` y con el id `$id`. Existe otro método, el `addInsert($id_etiqueta_anterior, $etiqueta, $id)`, que funciona exactamente igual que el `addCreate`, pero éste, en lugar de crear una etiqueta `$etiqueta` dentro de `$id_etiqueta_superior`, la crea ANTES de la etiqueta con el id `$id_etiqueta_anterior`. Pongamos un ejemplo.

Imagínate que tenemos en la página una etiqueta `<div>` con el id `"etiqueta1"`.

```
<div id="etiqueta1" />
```

Si aplicamos el método `addCreate("etiqueta1", "div", "etiqueta2")`, el resultado será éste:

```
<div id="etiqueta1">
  <div "etiqueta2" />
</div>
```

Pero si usamos en lugar de `addCreate`, `addInsert("etiqueta1", "div", "etiqueta2")`, el resultado sería éste otro:

```
<div id="etiqueta2" />
<div id="etiqueta1" />
```

Así que `addCreate` subordinaría la etiqueta que crea a la que indica en su primer argumento, y `addInsert` la pondría al mismo nivel.

De éstas dos, derivan otras, como puede ser `addInsertAfter($id_etiqueta_posterior, $etiqueta, $id)`, parecida al `addInsert` pero en lugar de crear la etiqueta ANTES de la que indica su primer argumento, la crea DESPUES.

Y derivan también:

- ✓ `addCreateInput($id_etiqueta_superior, $type, $nombre, $id)`
- ✓ `addInsertInput($id_etiqueta_anterior, $type, $nombre, $id)`
- ✓ `addInsertInputAfter($id_etiqueta_posterior, $type, $nombre, $id)`

para crear etiquetas del tipo `input`. A éstas se les aplican los atributos `type = $type` y `name = $nombre`.

Métodos de Eliminación

De métodos para eliminar una tag sólo hay uno, el `addRemove($id)` y tan sólo se le ha de pasar como argumento el id de la etiqueta a eliminar.

Métodos de Modificación

Los métodos de modificación permiten cambiar el contenido (texto) o un atributo de una etiqueta. En primer lugar tenemos el `addAssign($id, $atributo, $datos)`, que reemplaza todo el contenido de un atributo (`$atributo`) de una etiqueta (con el `id $id`) por el texto `$datos`. Así que si se quiere modificar el contenido de una capa `<div>` o un párrafo `<p>`, tan sólo has de pasar por `$id` el del párrafo o la capa, por `innerHTML` el `$atributo` y como texto nuevo el `$datos`. Veamos un ejemplo.

Tenemos un `div` con `id "etiqueta3"`.

```
<div id="etiqueta3" />
```

Si lo modificamos desde una función xajax con una respuesta a la que se le ha aplicado el método `addAssign("etiqueta3", "innerHTML", "Este es el contenido de la capa")`, entonces ésta capa se cambiará y será:

```
<div id="etiqueta3">Este es el contenido de la capa</div>
```

Y si le queremos poner la anchura (`width`) al 25%, llamaríamos a una función que devuelva una respuesta que haya sido modificada por el método `addAssign("etiqueta3", "style.width", "25%")`. El resultado es este.

```
<div id="etiqueta3" style="width:25%">Este es el contenido de la capa</div>
```

Hay otros métodos que no reemplazan sino que añaden texto al que ya hay actualmente. Se trata de los métodos `addAppend($id, $atributo, $datos)` y `addPrepend($id, $atributo, $datos)`. El método `addAppend` añade texto al fin del texto que ya hay en la etiqueta con `id $id` y `addPrepend` lo añade antes de éste.

El método `addReplace($id, $atributo, $texto a buscar, $texto a reemplazar)` permite, en el atributo `$atributo` de la etiqueta con el `id $id`, reemplazar el texto `$texto_a_buscar` por el texto `$texto a reemplazar`.

Y finalmente el método `addClear($id, $atributo)` borra el contenido de un atributo (`$atributo`) o del contenido (poniendo por `$atributo innerHTML`) de una etiqueta (con `id $id`). Es lo mismo que hacer un `addAssign($id, $atributo, "")`.

Que a nadie se le ocurra, bajo ningún concepto, utilizar una función xajax simplemente para cambiar el ancho o algún atributo de una etiqueta como lo hago yo en estos ejemplos. Esto se puede hacer perfectamente desde JavaScript y por supuesto no hace falta hacer una llamada al servidor para realizar una cosa como ésta. El servidor servirá para recibir información, y si se puede modificar algún atributo es simplemente para adecuar la página a la nueva información.

Métodos de Interacción con JavaScript

Ésta es una colección de métodos que ejecutan JavaScript. Empezaremos explicando el método `addScript($código_javascript)`, que ejecuta el código JavaScript que tiene como parámetro. El método `addScriptCall($función_javascript, $arg1, $arg2...)`, llama a una función JavaScript con el nombre `$function_javascript` y como argumentos los argumentos del segundo al último de éste método.

El método `addIncludeScript($fichero_js)` tiene como argumento la ruta de un fichero `js` y lo carga dentro del documento.

Los métodos `addAlert($advertencia)` y `addRedirect($pagina_nueva, $tiempo)` son exactamente lo mismo que `addScript("alert(\".$advertencia.\");")`, que alerta la `$advertencia` y que `addScript("setTimeout(window.location='\".$pagina_nueva.\"', \"$tiempo*1000.\");")` que redirecciona a la página `$pagina_nueva` al cabo de `$tiempo` segundos.

El método `addConfirmComands($num, $mensaje)` lanza un `prompt` con el `$mensaje`. Si el usuario presiona `Cancelar`, los siguientes `$num` comandos serán ignorados. Pongamos un ejemplo:

```
function prueba(){
// . . . // código que interactua con el servidor

$respuesta =new xajaxResponse();
$respuesta->addConfirmComands(2, "Quieres eliminar la etiqueta 4 y 5?");
$respuesta->addRemove("etiqueta4");
$respuesta->addRemove("etiqueta5");
$respuesta->addAppend("etiqueta6", "innerHTML", $alguna_variable);
```

Cuando el usuario reciba la respuesta le saldrá un `prompt` que le preguntará si quiere eliminar las `etiquetas 4 y 5`. Si responde `aceptar`, los dos siguientes comandos se ejecutarán y se eliminarán las dos etiquetas, si responde `cancelar`, los dos comandos serán ignorados. En cualquiera de los dos casos, en la `etiqueta 6` se escribirá el texto `$alguna_variable`.

Métodos de Adición de Eventos

El método `addEvent($id, $evento, $código_javascript)` añade un evento a una etiqueta con el `id $id`, y cuando se dispara ese evento, se ejecuta el `$código_javascript`. El método `addHandler($id_etiqueta, $evento, $función_javascript)` funciona de forma parecida pero en lugar de poner un código como tercer argumento has de ponerle una función JavaScript. Éste evento se puede eliminar con `addRemoveHandler($id_etiqueta, $evento, $función_javascript)`.

La respuesta XML de xAJax tendrá configurada por defecto la codificación "`utf-8`". Aún así, puedes cambiarla con un método del `xajaxResponse` llamado `setCharEncoding($codificación)`, si aún no la has cambiado al crear el objeto.

Por defecto, cuando usas un método de modificación donde añades texto donde hay etiquetas HTML, éstas etiquetas serán leídas por el navegador como tales (si añades texto con un método como éstos y en el texto hay una palabra entre etiquetas ` `, ésta palabra se verá en negrita en el navegador). Si se quiere evitar esto, se puede llamar al método `outputEntitiesOn()` para que las etiquetas se muestren como texto en el navegador. Para volver al estado anterior, usa el método `outputEntitiesOff()`. También tienes la posibilidad de cambiarla cuando creas el objeto.

Y finalmente, el método `loadXML($xml)` permite asignar a un objeto `xajaxResponse` el XML de otro objeto `xajaxResponse`.

XAJAX - EL OBJETO

Anteriormente hablábamos de las funciones PHP xAJax que permiten modificar la página web una vez ya cargada en el navegador. Éstas funciones, han de ser recogidas por un **objeto xAJax**.

El constructor de éste objeto tiene cuatro argumentos.

- ✓ `$pagina_peticion` – Página a la cual se llamará para hacer la petición. Por defecto es la página actual. Para cambiarla una vez creado el objeto, utiliza `setRequestURI($pagina_peticion)`.
- ✓ `$prefijo` – Prefijo que precederá al nombre de las funciones JavaScript. Por defecto "xajax". Para cambiarlo una vez construido el objeto, llama al método `setWrapperPrefix($prefijo)` pasándole como argumento el prefijo deseado.
- ✓ `$codificación` – Codificación que se usará para hacer las peticiones y respuestas. Por defecto "utf-8". Si quieres cambiarla después de haber construido el objeto, usa el método `setCharEncoding($codificación)` para cambiarla.
- ✓ `$debug` – Booleano que indica si saldrá la ventana `Xajax Debug Output`. Por defecto `false`. Para cambiarlo una vez construido el objeto, usa el método `debugOn()` para activarlo y `debugOff()` para desactivarlo.

Así que si creas un **objeto xajax** sin argumentos, sus variables cogerán los valores por defecto.

Una vez creado y configurado el objeto, pasaremos a decirle qué funciones podrán ser accesibles desde JavaScript (serán funciones en las que se cree el **objeto** `xajaxResponse`).

Si quieres poner en las respuestas y peticiones acentos o caracteres especiales, habrás de usar una codificación "iso-8859-1", pasándola como argumento al tercer argumento del constructor o como argumento del método `setCharEncoding("iso-8859-1")`. También sería interesante que usaras el método `decodeUTF8InputOn()` para pasar caracteres especiales en los argumentos de las funciones xajax.

Para registrar una función, hay que pasarla como argumento por el método `registerFunction($función, $tipoPeticion)`. La `$función` sería la función PHP que ya hemos creado, y el `$tipoPeticion` es un booleano; si es `true`, los datos se enviarán por `POST`, si es `false`, por `GET`, si no pones nada, por defecto será `POST`.

Una vez tenemos todas las funciones que queremos registrar, llamamos a la función `processRequests()`, necesaria para que nuestra aplicación xajax funcione.

Has de ir con mucho cuidado de no enviar nada al ordenador cliente sin antes haber llamado al método `processRequests()`. Lo primero que ha de haber en la página ha de ser `<?php` seguido de las funciones PHP y el **objeto xajax**.

Esto es todo lo del lado del servidor, en código sería esto:

```
<?php
// Antes de enviar nada hay que registrar todas las funciones, así que el texto ha de
// empezar así, sin ningún carácter antes

// Añadimos la librería
require_once('xajax/xajax.inc.php');

// Declaramos las funciones PHP
function funcion1 ($arg){
    // Aquí todo el código que interactúa con el servidor
    // ...
    $variable_cogida_antes=$arg;
    // Creamos una respuesta
    $respuesta=new xajaxResponse();

    // modificamos la respuesta con tantos métodos como queramos
    $respuesta->addCreate("body", "form", "form1");
    $respuesta->addCreateInput("form1", "text", "user", "user");
    $respuesta->addAssign("user", "value", $variable_cogida_antes);
    $respuesta->addInsertInput("user", "submit", "ok", "ok");
```



```

    $respuesta->addAssign("ok", "value", "OK");
    $respuesta->addHandler("form1", "onsubmit", "xajax_funcion2");
    // Tantos métodos como quieras!

    return $respuesta; // o return $respuesta->getXML(), como prefieras
}

function funcion2 (){

    // . . .

}

// Y tantas funciones como quieras!! ;)

// Creamos el objeto xajax
$xajax=new xajax(); // Todos los valores por defecto

//$xajax->debugOn(); // Activamos la ventana de Debug, si queremos

// Registramos todas las funciones
$xajax->registerFunction("funcion1"); // La respuesta se realizará por POST (default)
$xajax->registerFunction("funcion2", 0);
// La respuesta de la segunda función se realizará por GET (false en segundo argumento)

// Y procesamos la respuesta
$xajax->processRequests();
?>

```

Éste código no es funcional. Tan sólo me servirá para transmitirte la idea de una aplicación xajax: primero se añade la librería, después se configuran las funciones, éstas se registran y se llama al método `processRequests()`;

Has de tener en cuenta que no puedes registrar una función (llamar al método `registerFunction`) después de haber procesado la respuesta, así,

```

<?php
// . . .
$xajax->registerFunction(funcion1);
$xajax->processRequests();
$xajax->registerFunction(funcion2, 0);
$xajax->processRequests(); //Éste ya no funcionará
// . . .
?>

```

Del código anterior sólo sería operativa la primera función.

XAJAX - EN EL LADO CLIENTE

Una vez explicado lo anterior, ya podemos pasar a explicar cómo funciona el xajax en el lado del cliente. Resulta que el objeto xajax que hemos creado tiene un método que no hemos explicado antes. Se trata del `printJavascript($url, $archivo)` que antes de enviar la página al cliente escribe el código JavaScript necesario para hacer que todo funcione correctamente. En su primer argumento le habrás de pasar la carpeta donde tienes instalado el xajax, y si no has modificado la carpeta del zip, no hará falta que modifiques su segundo argumento.

Y ya está, ahora ya puedes llamar a una función del servidor que modificará la página sin recargarla tan sólo llamando a una función JavaScript. Aún así, tendrás que poner el prefijo `xajax` (o el que hayas marcado en el constructor de xajax) seguido del nombre de la función PHP.

Ejemplo

Esto es todo lo que voy a explicar sobre xAJax. Si te interesa el tema mírate [la documentación](#) que aunque está en inglés se puede entender perfectamente. Ahora vamos a poner todo lo explicado en práctica.

¿Nunca has visto en un formulario de registro en alguna web que al lado de el campo de texto de usuario hay un link que te permite comprobar, antes de enviar el formulario, si el nombre de usuario está libre o está ya en uso? De eso es de lo que trata el ejemplo.

Tenemos cuatro campos, el del usuario, dos de contraseñas (el segundo para comprobarla) y uno de e-mail. Todos ellos son fácilmente validables desde JavaScript; comprobando si el usuario ha introducido la misma contraseña en los dos campos, comprobando que en el e-mail hay una arroba, un punto, un nombre de dominio, que no hay ningún carácter especial... pero en ningún momento podrás comprobar tan sólo con JavaScript si el nombre de usuario está ya en la base de datos de usuarios de la web. Para eso se requiere AJAX, para hacer de puente entre el servidor y el cliente, y con xAJax lo podemos hacer mucho más fácil.

```
<?php
    require once('xajax/xajax.inc.php');

    function compruebaUser($user){
        $disponibilidad="$user est&aacute; disponible.";
        if($user=="LloX") $disponibilidad="$user no est&aacute; disponible.";
        $disponibilidad.=' <a href="'
onclick="xajax_compruebaUser(document.form1.user.value); return false;">Volver a
probar</a>.';
        $respuesta=new xajaxResponse();
        $respuesta->addAssign("disp", "innerHTML", $disponibilidad);
        return $respuesta;
    }
    $xajax=new xajax();
    $xajax->setCharEncoding("iso-8859-1");
    $xajax->decodeUTF8InputOn();
    $xajax->registerFunction("compruebaUser");
    $xajax->processRequests();
?>
```

Éste sería todo el código que se ejecutaría en la parte del servidor. Lo primero que hacemos es llamar a la librería xAJax, y luego escribimos una función llamada `compruebaUser` que comprobará si el usuario que se le ha pasado como argumento está disponible o no, y, en cualquier caso, devolver una respuesta XML que cambie el contenido de una capa llamada `"disp"`, que se encontrará situada debajo del campo de texto. Configuraríamos también el objeto xajax con la función `compruebaUser`. Le cambiaríamos la codificación para que se puedan usar caracteres especiales en las transferencias.

A continuación, seguiríamos con todo el código que se ejecutaría en el lado del cliente.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
    <title>Ejemplo de formularios con xAJax</title>

    <style type="text/css">
      body{background:#3399FF}
      #registro{
        width:360px;
        padding:10px;
        margin:15px;
        border: 2px solid #FFFF99;
        background-color: #3399FF;
      }
    </style>
  </head>
  <body>
    <div id="registro">
      <input type="text" value="Usuario" />
      <input type="password" value="Contraseña" />
      <input type="password" value="Repetir contraseña" />
      <input type="text" value="E-mail" />
      <input type="button" value="Registrar" />
      <div id="disp"></div>
    </div>
  </body>
</html>
```

```

    }
    #registro h1{
        font:bold 16px Arial, sans-serif;
        color:#FFFF99;
        padding:0 5px;
        border-bottom:solid #FFFF99 0.5px;
    }
    #registro label{
        font:bold 10px Geneva, Arial, Helvetica, sans-serif;
        color:#FFFF99;
        float:left;
        width:22%;
        text-align:right;
        padding:8px;
        margin-top:5px;
    }
    #registro span{
        font:bold 10px Geneva, Arial, Helvetica, sans-serif;
        color:#FFFF99;
        text-align:left;
        margin-left:24%;
        padding:8px;
    }
    #registro a{color:#FFFF99;}
    #registro a:hover{color:#FF0099;}
    #registro input{
        border:#FFFF99 solid 0.5px;
        background:#3399FF;
        color: #FFFF99;
        margin-top:9px;
    }
    #registro br{clear: left;}
    .enviar{margin:3px 0 0 83px;}
</style>

<?php $xajax->printJavascript("xajax"); ?>

</head>

```

Tan sólo sería aplicarle estilo al formulario que vendrá ahora y una llamada al método `printJavascript` para escribir automáticamente allí todo el código que permitirá que funcione xajax.

También en ésta parte del código podríamos poner un script con funciones para validar los elementos del formulario, aunque para economizar espacio y porque ya no es un tema relacionado con xajax, no lo pongo.

Y finalmente, éste sería el cuerpo de la página con el formulario.

```

<body>
    <div id="registro">

        <form id="form1" name="form1" method="post" action="">
            <h1>Registro usuarios.</h1>
            <p>
                <label>Usuario</label>
                <input type="text" name="user" />
                <br />
                <span id="disp">
                    <a href="" onclick="xajax_compruebaUser(document.form1.user.value);
return false;">Comprobar disponibilidad.</a>
                </span>
                <br />
                <label>Contrase&ntilde;a</label>
                <input type="password" name="password" />
                <br />
                <label>Repite Contrase&ntilde;a</label>
                <input type="password" name="password2" />
                <br />
                <label>Correo Electr&oacute;nico</label>
                <input type="text" name="email" />
            </p>
        </form>
    </div>

```

```
        <br />
        <input type="submit" name="Submit" value="Enviar" class="enviar" />
    </p>
</form>

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

Ahí tendríamos los cuatro campos de texto con un link en el primero para hacer una comprobación por xajax. Al hacer clic en el link, se llamará a la función JavaScript `xajax compruebaUser`, que por la magia de xajax llamará a una función PHP `xajax`, la función `compruebaUser`, que le devolverá una respuesta en XML. La misma función recoge la respuesta y la interpreta, modificando el contenido de la capa donde está el enlace y mostrando la disponibilidad del usuario.

Página simple con xajax y PHP

Veamos ahora como realizar una página que utilice xajax, para ejecutar una sencilla función PHP como respuesta a una acción del usuario. El ejemplo es relativamente sencillo, incluso lo podemos hacer en pocos pasos, como una receta. Luego se podrá complicar todo lo necesario para realizar acciones más complejas.

1) Incluir con PHP el archivo donde está la clase xajax

```
//incluimos la clase ajax
require ('xajax/xajax.inc.php');
```

2) Creamos una instancia de un objeto de la clase xajax

```
//instanciamos el objeto de la clase xajax
$xajax = new xajax();
```

3) Escribimos una función en PHP, que luego llamaremos con por medio de ajax

Esta función es todo lo complicado que se requiera. Realizará acciones del lado del servidor y por tanto puede acceder a bases de datos, abrir ficheros o cualquier cosa que se nos ocurra. Luego en la propia función realizaremos una instancia de un objeto `AjaxResponse`, que utilizaremos para mostrar resultados en la página.

```
function si_no($entrada){
    if ($entrada=="true"){
        $salida = "Marcado";
    }else{
        $salida = "No marcado";
    }

    //instanciamos el objeto para generar la respuesta con ajax
    $respuesta = new xajaxResponse();
    //escribimos en la capa con id="respuesta" el texto que aparece en $salida
    $respuesta->addAssign("respuesta","innerHTML",$salida);

    //tenemos que devolver la instanciación del objeto xajaxResponse
    return $respuesta;
}
```

El objeto `xajaxResponse()` sirve para realizar acciones en la página sin tener que recargar el documento. Dispone de varios métodos o funciones, como por ejemplo `addAssign()` que sirve para asignar un valor a una propiedad de un elemento de la página. En este caso se asigna el valor contenido en la variable `$salida` al `innerHTML` de la capa `"respuesta"`, con lo que se altera el texto contenido en esa capa.

4) Asociamos la función PHP al objeto xajax

```
//asociamos la función creada anteriormente al objeto xajax
$xajax->registerFunction("si_no");
```

Esta asociación permitirá ejecutar la función PHP desde una llamada a una función Javascript.

5) Antes de enviar cualquier contenido a la página, tenemos que ejecutar un método del objeto xajax para procesar las peticiones del que puedan llegar a la página.

```
//El objeto xajax tiene que procesar cualquier petición
$xajax->processRequests();
```

Insistimos, esta llamada al método se tiene que hacer antes de escribir ningún contenido dentro del código de la página, es decir, antes que llegue al cliente ningún carácter de código HTML.

6) Escribir el código javascript necesario para procesar las llamadas a ajax.

```
//En el <head> indicamos al objeto xajax se encargue de generar el javascript necesario
$xajax->printJavascript("xajax/");
```

Lo ideal es hacer esta llamada al método `printJavascript()` dentro del `<head>` de la página.

Si nos fijamos, el método recibe un parámetro, que es la ruta relativa para acceder al directorio donde están los archivos xajax descomprimidos.

7) Podemos hacer llamadas a las funciones PHP en cualquier lugar del código, como respuesta a las acciones del usuario con javascript.

```
<input type="checkbox" name="si" value="1"
onclick="xajax_si_no(document.formulario.si.checked) ">
```

Como podemos ver, desde un elemento de la página, como en este caso una casilla de verificación, al cambiar su estado, se llama a una función javascript para ejecutar la función PHP escrita anteriormente. Es decir, al pulsar el checkbox se desencadena el evento `onchange` y con él se llama a la función `xajax_si_no()` enviando como parámetro el estado (chequeado o no) de la casilla de verificación.

Podemos ver el código del ejemplo completo a continuación, pero tener en cuenta que para que funcione tiene que disponer del código de la clase xajax, que en este caso debe estar en un subdirectorío que cuelgue del directorio donde está el archivo del ejemplo.

```
<?
//incluimos la clase ajax
require ('xajax/xajax.inc.php');

//instanciamos el objeto de la clase xajax
$xajax = new xajax();

function si_no($entrada){
    if ($entrada=="true"){
        $salida = "Marcado";
    }else{
        $salida = "No marcado";
    }

    //instanciamos el objeto para generar la respuesta con ajax
    $respuesta = new xajaxResponse();
    //escribimos en la capa con id="respuesta" el texto que aparece en $salida
    $respuesta->addAssign("respuesta", "innerHTML", $salida);

    //tenemos que devolver la instanciación del objeto xajaxResponse
    return $respuesta;
}

//asociamos la función creada anteriormente al objeto xajax
$xajax->registerFunction("si no");

//El objeto xajax tiene que procesar cualquier petición
$xajax->processRequests();
?>

<html>
<head>
    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
    <META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=ISO-8859-1">
    <title>Si / No en Ajax</title>
    <?
    //En el <head> indicamos al objeto xajax se encargue de generar el javascript
necesario
    $xajax->printJavascript("xajax/");
    ?>
</head>
```

```
<body>
<div id="respuesta"></div>
<form name="formulario">
<input type="checkbox" name="si" value="1"
onclick="xajax_si_no(document.formulario.si.checked)">
</form>

<script type="text/javascript">
  xajax si_no(document.formulario.si.checked); //Llamando inicialmente a la función
xajax_si_no inicializamos el valor de la capa con la respuesta
</script>
</body>
</html>
```

Recibir formulario con Ajax e insertar información en base de datos.

Estamos viendo algunos ejemplos de Ajax utilizado junto con PHP para crear aplicaciones web avanzadas, ayudándonos de la clase xajax, que facilita bastante la programación.

Ahora vamos a ver cómo procesar un formulario en una página web sin que se tenga que recargar la página, es decir, enviar al servidor los datos adjuntados por el usuario en el formulario, procesarlos con PHP y luego devolver una salida al usuario dependiendo de los datos introducidos. Todo sin cambiar la página en la que estaba el usuario.

Aprendimos anteriormente a llamar a funciones PHP desde Javascript. Ahora lo que tenemos que hacer es llamar a una función PHP para que procese un formulario. La llamada la seguiremos haciendo mediante Javascript cuando el usuario pulse el botón de enviar. La particularidad de este caso de envío y procesamiento de formularios con Ajax es la manera de enviar y recoger los datos del formulario.

Enviar los datos del formulario a PHP con xajax

Podemos ver aquí el código del formulario:

```
<div id="mensaje">
  <form id="formulario">
    Nombre: <input type="text" name="nombre">
    <br>
    Apellidos: <input type="text" name="apellidos">
    <br>
    <input type="button" value="Enviar"
onclick="xajax_procesar_formulario(xajax.getFormValues('formulario'))">
  </form>
</div>
```

Como vemos, tenemos un par de campos de texto y un botón, que se encargará lanzar el proceso para enviar y procesar los datos. Para ello, el botón tiene un comando para el evento `onclick`, que trataremos de explicar.

Ahora con xajax vamos a disponer de un método Javascript llamado `xajax.getFormValues(ID FORMULARIO)`, que recibe el identificador del formulario que se desean obtener los datos. Esta función nos sirve generar los datos del formulario que debemos enviar para su procesamiento. En nuestro caso el formulario tiene el `id="formulario"`, así que lo invocamos:

```
xajax.getFormValues('formulario')
```

Los datos del formulario los tenemos que pasar a la función que se encarga de procesarlos, que es una función que hemos escrito con PHP y luego hemos registrado en el objeto xajax. Veremos esa función a continuación.

Recibir los datos del formulario enviado con xajax

Desde PHP estaríamos acostumbrados a recoger los datos de un formulario con `$ _POST` o `$ _GET`, pero ahora lo vamos a tener que hacer de una manera distinta.

Como hemos visto antes, a la función que procesa los el formulario se le envía un parámetro con los datos. Pues ese mismo parámetro es el que tenemos que utilizar para recibir los datos del formulario. Los datos del formulario nos llegan como un array asociativo, por tanto, cuando programemos la función en PHP, el método de trabajo será similar al que utilizaríamos con `$ _POST` y `$ _GET`, que también son arrays asociativos.

```
function procesar_formulario($form_entrada){
    $salida = "Gracias por enviarnos tus datos. Hemos procesado esto:<p>";
    $salida .= "Nombre: " . $form_entrada["nombre"];
    $salida .= "<br>Apellidos: " . $form_entrada["apellidos"];

    //instanciamos el objeto para generar la respuesta con ajax
    $respuesta = new xajaxResponse();
    //escribimos en la capa con id="respuesta" el texto que aparece en $salida
    $respuesta->addAssign("mensaje", "innerHTML", $salida);

    //tenemos que devolver la instanciación del objeto xajaxResponse
    return $respuesta;
}
```

La función que procesa el formulario se llama `procesar_formulario()` y recibe un parámetro que es el array asociativo con los datos del formulario. Esa función compone una salida y luego la devuelve con Ajax.

Para ello crea un objeto de la clase `xajaxResponse` y con el método `addAssign("mensaje", "innerHTML", $salida)` consigue que en la capa con `id="mensaje"` se muestre la salida del procesamiento del formulario.

No hay que olvidarse de registrar la función en el objeto xajax

```
//registramos la función creada anteriormente al objeto xajax
$xajax->registerFunction("procesar_formulario");
```

El código completo de este ejemplo es el siguiente:

```
<?
//incluimos la clase ajax
require ('xajax/xajax.inc.php');

//instanciamos el objeto de la clase xajax
$xajax = new xajax();

function procesar_formulario($form_entrada){
    $salida = "Gracias por enviarnos tus datos. Hemos procesado esto:<p>";
    $salida .= "Nombre: " . $form_entrada["nombre"];
    $salida .= "<br>Apellidos: " . $form_entrada["apellidos"];

    //instanciamos el objeto para generar la respuesta con ajax
    $respuesta = new xajaxResponse();
    //escribimos en la capa con id="respuesta" el texto que aparece en $salida
    $respuesta->addAssign("mensaje", "innerHTML", $salida);

    //tenemos que devolver la instanciación del objeto xajaxResponse
    return $respuesta;
}
//registramos la función creada anteriormente al objeto xajax
$xajax->registerFunction("procesar_formulario");

//El objeto xajax tiene que procesar cualquier petición
$xajax->processRequest();
?>
<html>
<head>
```



```

<title>Enviar y procesar un formulario con Ajax y PHP</title>
<?
    //En el <head> indicamos al objeto xajax se encargue de generar el javascript
    //necesario
    $xajax->printJavascript("xajax/");
?>
</head>

<body>
    <h1>Recibir y procesar formulario con Ajax y PHP</h1>
    <div id="mensaje">
        <form id="formulario">
            Nombre: <input type="text" name="nombre">
            <br>
            Apellidos: <input type="text" name="apellidos">
            <br>
            <input type="button" value="Enviar"
onclick="xajax_procesar_formulario(xajax.getFormValues('formulario'))">
        </form>
    </div>

</body>
</html>

```

Si necesitásemos insertar estos datos en una base de datos no variaría mucho el ejemplo.

Envío de formulario e insertar la información en una base de datos MySQL.

En este artículo vamos a resolver la duda de un usuario, que quería insertar en una base de datos la información recibida de un formulario con Ajax. En realidad es un tema que no revierte ninguna complicación, si ya conocemos el modo de trabajo de PHP con bases de datos, pues no varía nada que estemos realizando las acciones a través de Ajax. Pero bueno, puede ser de utilidad explicarlo.

Si quisiéramos, podríamos insertar la información recibida por el formulario en una base de datos. Esto sólo implicaría un pequeño cambio en la función `procesar formulario()`, para que realice el `insert`. En lugar de mostrar los datos por pantalla como hace en el ejemplo anterior, tendría que generar una sentencia SQL con el `insert` y ejecutarla.

Tendríamos también que realizar una conexión con la base de datos donde queremos hacer el `insert`. Esta conexión podríamos hacerla dentro de la misma función o fuera. En este pequeño código de la función `procesar formulario()` se muestra como podría ser el proceso de inserción de la información en una base de datos MySQL:

```

function procesar_formulario($form_entrada){
    $connectid = mysql_connect("localhost", "root", "");
    mysql_select_db("nombre_base_datos",$connectid);
    $ssql = "insert into pais (nombre_pais) values ('" . $form_entrada["nombre"] . "')";

    if (mysql_query($ssql)){
        $salida = "Insertado correctamente";
    }else{
        $salida = "No se ha insertado. Este es el error: " . mysql_error();
    }

    //instanciamos el objeto para generar la respuesta con ajax
    $respuesta = new xajaxResponse();
    //escribimos en la capa con id="respuesta" el texto que aparece en $salida
    $respuesta->addAssign("mensaje","innerHTML",$salida);

    //tenemos que devolver la instanciación del objeto xajaxResponse
    return $respuesta;
}

```

Pero la sentencia de conexión con la base de datos podría estar en otro lugar del código de la página. En un supuesto que nuestra página realice accesos a base de datos en diversos lugares del código, nos convendría realizar una conexión a la base de datos de manera global, que podamos utilizar desde cualquier parte del código.

A continuación se muestra el ejemplo completo, de enviar datos de un formulario por Ajax e insertar el contenido en una base de datos MySQL. En este caso hemos hecho una variación en el código para que la conexión a la base de datos se realice como variable global a la página y no local a la función, así podríamos utilizar esa misma conexión en otros lugares del código PHP de la página.

```
<?
//incluimos la clase ajax
require ('xajax/xajax.inc.php');

//instanciamos el objeto de la clase xajax
$xajax = new xajax();

$connectid = mysql connect("localhost", "root", "");
mysql select db("guiarte backup",$connectid);

function procesar_formulario($form_entrada){
    $sssql = "insert into pais (nombre_pais) values ('" . $form_entrada["nombre"] .
    "')";

    if (mysql query($sssql)){
        $salida = "Insertado correctamente";
    }else{
        $salida = "No se ha insertado. Este es el error: " . mysql_error();
    }

    //instanciamos el objeto para generar la respuesta con ajax
    $respuesta = new xajaxResponse();
    //escribimos en la capa con id="respuesta" el texto que aparece en $salida
    $respuesta->addAssign("mensaje","innerHTML",$salida);

    //tenemos que devolver la instanciación del objeto xajaxResponse
    return $respuesta;
}
//registramos la función creada anteriormente al objeto xajax
$xajax->registerFunction("procesar formulario");

//El objeto xajax tiene que procesar cualquier petición
$xajax->processRequests();
?>
<html>
<head>
<title>Enviar y procesar un formulario con Ajax y PHP</title>
<?
    //En el <head> indicamos al objeto xajax se encargue de generar el javascript
    //necesario
    $xajax->printJavascript("xajax/");
?>
</head>

<body>
<h1>Recibir y procesar formulario con Ajax y PHP</h1>
<div id="mensaje">
    <form id="formulario">
        Nombre de país: <input type="text" name="nombre">
        <br>
        <input type="button" value="Enviar"
onclick="xajax_procesar_formulario(xajax.getFormValues('formulario'))">
    </form>
    </div>
</body>
</html>
```

Procesar formulario Ajax con validación

Estuvimos haciendo en un artículo anterior una página que envía un formulario con PHP y Ajax y lo procesa, devolviendo los resultados sin que se tenga que recargar la página. Ahora vamos a complicar ese ejemplo, creando un formulario que tiene distintas validaciones en el servidor. El formulario no se procesa hasta que no se valide correctamente la información y se muestran los posibles errores dentro de la propia página.

El formulario y la zona de mensajes

El formulario que hemos utilizado es muy parecido al anterior. Sólo le hemos añadido un campo más, de tipo `checkbox`, para realizar una validación un poco más compleja. Además, hemos incorporado una capa más para mostrar mensajes.

```
<div id="mensaje">
    Rellena los datos de este formulario y pulsa "Enviar"
</div>
<br />
<div id="capaformulario">
    <form id="formulario">
        Nombre: <input type="text" name="nombre" />
        <br />
        Apellidos: <input type="text" name="apellidos" />
        <br />
        <input type="checkbox" name="acepto" value="1" /> Acepto los términos y
condiciones ;)
        <br />
        <input
                                type="button"
                                value="Enviar"
onclick="xajax_procesar_formulario(xajax.getFormValues('formulario'))" />
    </form>
</div>
```

La capa de mensajes nos servirá para mostrar textos, como errores de validación, cuando se produzcan. El formulario está en una capa independiente, que sólo actualizaremos si finalmente se realiza el procesamiento de sus datos.

Ahora veamos la función PHP que realizará la validación. Si se produjeron errores en la validación actualizará la capa "`mensaje`" para mostrar el error. Si todo es correcto, procesará el formulario, mostrará un mensaje de confirmación en la capa "`mensaje`" y el resultado de procesar el formulario en la capa "`capaformulario`". Es una función un poco larga:

```
function procesar_formulario($form entrada){
    //creo el xajaxResponse para generar una salida
    $respuesta = new xajaxResponse('ISO-8859-1');

    //validación
    $error_form = "";
    if ($form_entrada["nombre"] == "")
        $error_form = "Debes escribir tu nombre";
    elseif ($form_entrada["apellidos"] == "")
        $error_form = "Debes escribir tus apellidos";
    elseif (!isset($form_entrada["acepto"]))
        $error_form = "Debes aceptar los términos y condiciones";

    //compruebo resultado de la validación
    if ($error_form != ""){
        //Hubo un error en el formulario
        //en la capa donde se muestran mensajes, muestro el error
        $respuesta->addAssign("mensaje","innerHTML","<span
style='color:red;'>$error_form</span>");
    }else{
        //es que no hubo error en el formulario
        $salida = "Hemos procesado esto:<p>";
        $salida .= "Nombre: " . $form_entrada["nombre"];
        $salida .= "<br>Apellidos: " . $form_entrada["apellidos"];
```

```
//mostramos en capa mensaje el texto que está todo correcto
$respuesta->addAssign("mensaje","innerHTML","<span style='color:blue;'>Todo
correcto... Muchas gracias!</span>");
//escribimos en la capa con id="capaformulario" el texto que aparece en $salida
$respuesta->addAssign("capaformulario","innerHTML",$salida);

//tenemos que devolver la instanciación del objeto xmlhttpResponse
}
return $respuesta;
}
```

La función está comentada, por lo que se podrá entender. Lo importante es fijarse en cómo se instancia el objeto de la clase `xajaxResponse` y cómo se invocan los distintos métodos para actualizar el contenido de las capas "mensaje" y "capaformulario".

Luego está el tema de las validaciones y la comprobación para saber si hubo un error o no en los datos. Este podría servir de esquema general, pero tema de las validaciones cada persona lo tendrá que implementar según sus necesidades.

Solución para los acentos en xajax

También vamos a ver un par de detalles acerca de los acentos en Ajax, que nos resolverán más de un dolor de cabeza. Posiblemente hayamos podido comprobar en este ejemplo, o en otros anteriores de Ajax, que los acentos muchas veces se ven mal, convertidos en algún carácter raro. Esto lo podemos solucionar de varias maneras con xajax, y nosotros hemos implementado una de ellas en este ejemplo.

Primero, cuando se crea la instancia del objeto xajax, podemos decirle con qué juego de caracteres queremos trabajar. Y además, podemos decirle que cualquier cadena que nos envíen por `POST` o `GET` se convierta automáticamente al juego de caracteres que estamos usando.

```
//instanciamos el objeto de la clase xajax
$xajax = new xajax();
$xajax->setCharEncoding('ISO-8859-1');
$xajax->decodeUTF8InputOn();
```

Luego, cuando hacemos las instancias del objeto de la clase `xajaxResponse` para generar la salida, también tenemos que indicar en qué juego de caracteres estamos trabajando, si no podría dar problemas.

```
$respuesta = new xajaxResponse('ISO-8859-1');
```

Esto se hace en la función PHP que procesa los datos y genera la salida. Ya habíamos visto anteriormente el código de esta función.

Así quedaría el código completo de este ejemplo:

```
<?
//incluimos la clase ajax
require ('xajax/xajax.inc.php');

//instanciamos el objeto de la clase xajax
$xajax = new xajax();
$xajax->setCharEncoding('ISO-8859-1');
$xajax->decodeUTF8InputOn();

function procesar_formulario($form_entrada){
    //creo el xajaxResponse para generar una salida
    $respuesta = new xajaxResponse('ISO-8859-1');
```

```

//validación
$error_form = "";
if ($form_entrada["nombre"] == "")
    $error_form = "Debes escribir tu nombre";
elseif ($form_entrada["apellidos"] == "")
    $error_form = "Debes escribir tus apellidos";
elseif (!isset($form_entrada["acepto"]))
    $error_form = "Debes aceptar los términos y condiciones";

//compruebo resultado de la validación
if ($error_form != ""){
    //Hubo un error en el formulario
    //en la capa donde se muestran mensajes, muestro el error
    $respuesta->addAssign("mensaje","innerHTML","<span
style='color:red;'>$error_form</span>");
}else{
    //es que no hubo error en el formulario
    $salida = "Hemos procesado esto:<p>";
    $salida .= "Nombre: " . $form_entrada["nombre"];
    $salida .= "<br>Apellidos: " . $form_entrada["apellidos"];

    //mostramos en capa mensaje el texto que está todo correcto
    $respuesta->addAssign("mensaje","innerHTML","<span
style='color:blue;'>Todo
correcto... Muchas gracias!</span>");
    //escribimos en la capa con id="capaformulario" el texto que aparece en $salida
    $respuesta->addAssign("capaformulario","innerHTML",$salida);

    //tenemos que devolver la instanciación del objeto xajaxResponse
}
return $respuesta;
}
//registramos la función creada anteriormente al objeto xajax
$xajax->registerFunction("procesar_formulario");

//El objeto xajax tiene que procesar cualquier petición
$xajax->processRequests();
?>
<html>
<head>

    <title>Enviar y procesar un formulario con Ajax y PHP</title>
    <?
    //En el <head> indicamos al objeto xajax se encargue de generar el javascript
necesario
    $xajax->printJavascript("xajax/");
    ?>
</head>

<body>
<h1>Recibir y procesar formulario con Ajax y PHP</h1>
<div id="mensaje">
Rellena los datos de este formulario y pulsa "Enviar"
</div>
<br />
<div id="capaformulario">
<form id="formulario">
Nombre: <input type="text" name="nombre" />
<br />
Apellidos: <input type="text" name="apellidos" />
<br />
<input type="checkbox" name="acepto" value="1" /> Acepto los términos y condiciones ;)
<br />
<input
type="button"
value="Enviar"
onclick="xajax_procesar_formulario(xajax.getFormValues('formulario'))" />
</form>
</div>

</body>
</html>

```

Select combinado

Un ejemplo típico de las prestaciones de Ajax es la creación de `selects` combinados, es decir, una estructura de dos `selects`, donde uno tiene las posibles opciones en función de lo que se haya elegido en el otro. Veremos cómo hacer este sistema en PHP y Ajax, con la ayuda de la librería xajax, que nos facilitará bastante las cosas.

Para empezar veremos el formulario inicial con el primer `select` y el segundo sin opciones.

```
<form name="formulario">
  Provincia:
  <br>
  <select
    name="provincia"
    onchange="xajax_generar_select(document.formulario.provincia.options[document.formulario.provincia.selectedIndex].value)">
    <option value="999">Selecciona provincia</option>
    <option value=0>Madrid</option>
    <option value=1>Valencia</option>
    <option value=2>Barcelona</option>
    <option value=3>León</option>
  </select>
  <br>
  <br>
  Población:
  <div id="seleccombinado">
    <select name="poblaciones">
      <option value=0>Elegir provincia</option>
    </select>
  </div>
</form>
```

Vemos que se tiene dos campos `select`, el primero para las provincias y el segundo para las poblaciones. El primer campo tiene todas las opciones posibles. El segundo `select` inicialmente no tiene ninguna opción, porque estas se deben incluir en función de la provincia escogida en el primer campo. Vemos que el segundo `select` está metido en una capa con `id="seleccombinado"`, que actualizaremos luego con Ajax.

Además, hay que fijarse en el atributo `onchange` del primer `select`, que llama con Ajax, por medio de xajax, a la función PHP que se encargará de generar las opciones del segundo `select`.

Ahora vamos a ver una función PHP que generaría el código de un `select` en función de un parámetro que recibirá: la provincia. Con ese identificador de provincia generará el código del `select` con todas las poblaciones de esa provincia.

```
function select_combinado($id_provincia){
  //función para crear el select combinado
  //debe extraer las opciones de un select a partir de un parámetro

  //generamos unos arrays con distintas poblaciones de varias provincias
  //estos valores en un caso práctico seguramente se extraerán de base de datos
  //no habría que cargar todos en memoria, sólo hacer el select de las poblaciones de
  la provincia deseada
  $madrid = array("Madrid", "Las Rozas", "Móstoles", "San Sebastián de los Reyes");
  $valencia = array("Valencia", "La Eliana", "Paterna", "Cullera");
  $barcelona = array("Barcelona", "Badalona");
  $leon = array ("León", "Astorga", "Villamejil");
  $poblaciones = array($madrid, $valencia, $barcelona, $leon);

  //creo las distintas opciones del select
  $nuevo_select = "<select name='poblaciones'>";

  for ($i=0; $i<count($poblaciones[$id_provincia]); $i++){
    $nuevo_select .= '<option value="' . $i . '">' . $poblaciones[$id_provincia][$i] .
    '</option>';
  }
}
```

```

    $nuevo_select .= "</select>";
    return $nuevo_select;
}

```

La función anterior tiene poco de interés para lo que es el manejo de Ajax. Aquí hemos creado unos arrays para almacenar las poblaciones de las distintas provincias, pero en nuestras aplicaciones posiblemente tengamos las poblaciones en una base de datos. En ese caso lo que tendríamos que hacer es simplemente una consulta y recorrer un conjunto de registros.

En definitiva, la función recibe un identificador de provincia, que se utiliza para recorrer el array asociado a la provincia y generar un campo `select` con una opción por cada población.

Ahora vamos a ver la función que hace uso de xajax para procesar y actualizar la página con Ajax para cambiar las opciones del segundo `select`.

```

function generar_select($cod_provincia){
    //instanciamos el objeto para generar la respuesta con ajax
    $respuesta = new xajaxResponse('ISO-8859-1');

    if ($cod_provincia==999){
        //escribimos el select de poblaciones vacío
        $nuevo_select = '<select name="poblaciones">
                        <option value=0>Elegir provincia</option>
                        </select>
                        ';
    }else{
        $nuevo_select = select_combinado($cod_provincia);
    }
    //escribimos en la capa con id="seleccombinado"
    $respuesta->addAssign("seleccombinado", "innerHTML", $nuevo_select);

    //tenemos que devolver la instancia del objeto xajaxResponse
    return $respuesta;
}

```

Lo primero es instanciar un objeto de la clase `xajaxResponse` para generar la respuesta. Como se puede ver, recibimos el código de la provincia como parámetro. Comprobamos si ese código de la provincia es 999, porque es un caso especial (no se ha seleccionado ninguna provincia) y tenemos que generar el `select` de provincias vacío. En caso que el código de la provincia sea otra cosa entonces se lo pasamos a la función `select_combinado()`, vista anteriormente, para generar el `select` con las poblaciones de la provincia dada.

Para acabar, escribimos en la capa con `id="seleccombinado"` la cadena con el `select` que hemos generado. Para escribirlo utilizamos el método `addAssign()` del objeto de la clase `xajaxResponse` que ya conocíamos de anteriores ejercicios.

El código completo es el siguiente:

```

<?
//incluimos la clase ajax
require ('xajax/xajax.inc.php');

//instanciamos el objeto de la clase xajax
$xajax = new xajax();
$xajax->setCharEncoding('ISO-8859-1');
$xajax->decodeUTF8InputOn();

function select_combinado($id_provincia){
    //función para crear el select combinado
    //debe extraer las opciones de un select a partir de un parámetro

    //generamos unos arrays con distintas poblaciones de varias provincias
    //estos valores en un caso práctico seguramente se extraerán de base de datos

```

```

//no habría que cargar todos en memoria, sólo hacer el select de las poblaciones de
la provincia deseada
$madrid = array("Madrid", "Las Rozas", "Móstoles", "San Sebastián de los Reyes");
$valencia = array("Valencia", "La Eliana", "Paterna", "Cullera");
$barcelona = array("Barcelona", "Badalona");
$leon = array("León", "Astorga", "Villamejil");
$poblaciones = array($madrid, $valencia, $barcelona, $leon);

//creo las distintas opciones del select
$nuevo select = "<select name='poblaciones'>";

for ($i=0; $i<count($poblaciones[$id_provincia]); $i++){
//for ($i=0; $i<2; $i++){
    $nuevo select .= '<option value="' . $i . '">' . $poblaciones[$id_provincia][$i] .
'</option>';
}
$nuevo_select .= "</select>";
return $nuevo_select;
}

function generar select($cod_provincia){
//instanciamos el objeto para generar la respuesta con ajax
$respuesta = new xajaxResponse('ISO-8859-1');

if ($cod_provincia==999){
//escribimos el select de poblaciones vacío
$nuevo select = '<select name="poblaciones">
<option value=0>Elegir provincia</option>
</select>
';
}else{
    $nuevo select = select combinado($cod_provincia);
}
//escribimos en la capa con id="seleccombinado"
$respuesta->addAssign("seleccombinado", "innerHTML", $nuevo_select);

//tenemos que devolver la instanciación del objeto xajaxResponse
return $respuesta;
}

//asociamos la función creada anteriormente al objeto xajax
$xajax->registerFunction("generar select");

//El objeto xajax tiene que procesar cualquier petición
$xajax->processRequests();
?>

<html>
<head>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=ISO-8859-1">
<title>Validar usuario en Ajax</title>
<?
//En el <head> indicamos al objeto xajax se encargue de generar el javascript
necesario
$xajax->printJavascript("xajax/");
?>
</head>

<body>

<form name="formulario">
Provincia:
<br>
<select name="provincia"
onchange="xajax generar select(document.formulario.provincia.options[document.formulario
.provincia.selectedIndex].value)">
<option value="999">Selecciona provincia</option>
<option value=0>Madrid</option>
<option value=1>Valencia</option>
<option value=2>Barcelona</option>
<option value=3>León</option>
</select>
<br>
<br>
Población: <div id="seleccombinado">

```



```
<select name="poblaciones">
<option value=0>Elegir provincia</option>
</select>
</div>
</form>
</body>
</html>
```

Comprobar si un usuario es válido

En este caso vamos a implementar una utilidad típica de los formularios de registro de usuarios en una web: validar un nombre de usuario. Cuando un visitante escribe un nombre de usuario en un formulario tenemos que comprobar si el nombre de usuario es válido y no ha sido repetido por otra persona que se registró anteriormente. Esto se puede hacer con Ajax de una manera muy usable para el visitante, de modo que se pueda comprobar el usuario antes de enviar el formulario para su procesamiento y sin que tenga que recargarse la página.

El ejemplo en si es sencillo, pero requerirá de realizar unos cuantos pasos. Primero vamos a ver el formulario donde el usuario escribirá el nombre de usuario.

```
<form name="formulario">
  <input type="text" name="usuario" size="20"> <div id="mensaje"></div>
  <br>
  <input type="button" value="Comprobar validez de usuario"
  onclick="javascript:xajax_validar_usuario(document.formulario.usuario.value)">
</form>
```

Tiene un campo de texto donde se debe escribir el nombre de usuario. Una capa con `id="mensaje"` donde mostraremos mensajes de error o validez, y un botón que habrá que pulsar para que se llame a la función que se encargará de comprobar el correo.

Veamos ahora un par de funciones PHP que utilizaremos para las validaciones:

```
function comprobar_permitidos($cadena){
    $permitidos = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789- ";
    for ($i=0; $i<strlen($cadena); $i++){
        if (strpos($permitidos, substr($cadena,$i,1))===false){
            //no es válido;
            return false;
        }
    }
    //si estoy aqui es que todos los caracteres son validos
    return true;
}
```

Esta función comprueba si los caracteres de un nombre de usuario son válidos. Sólo se permiten alfanuméricos y el signo "-".

```
function comprobar_repetidos($cadena){
    //esta función comprueba si se ha repetido un nombre de usuario
    //se supone que aquí se debería hacer una búsqueda en base de datos para ver si hay
    repetidos
    //nosotros para este ejemplo no vamos a conectar con base de datos
    //simplemente comprobamos si la cadena es igual a unos valores literales
    if ($cadena == "pepe" || $cadena == "jose" || $cadena == "juan"){
        return false;
    }
    return true;
}
```

Esta otra función realiza una comprobación para ver si un usuario está repetido anteriormente. Lo lógico, como ya aparece comentado en el código de la propia función, es que hubiéramos

realizado un acceso a base de datos para comprobar si el usuario está o no ya en uso en la base de datos. En este ejemplo, sin embargo, sólo hemos comprobado si el usuario es igual a los valores "pepe", "jose" y "juan". Será suficiente para por el momento, porque lo que nos interesa es entender cómo trabajar con xajax.

Ahora vamos con lo importante, que es la función PHP que se encargará de comprobar la validez de un usuario y mostrar con Ajax los mensajes correspondientes, según el usuario sea válido o no.

```
function validar_usuario($entrada){
    //instanciamos el objeto para generar la respuesta con ajax
    $respuesta = new xajaxResponse('ISO-8859-1');

    if ($entrada == ""){
        //escribimos en la capa con id="mensaje" que no se ha escrito nombre de usuario
        $respuesta->addAssign("mensaje","innerHTML","Debes escribir algo como nombre de usuario");
        //Cambiamos a rojo el color del texto de la capa mensaje
        $respuesta->addAssign("mensaje","style.color","red");
    }elseif (!comprobar_permitidos($entrada)){
        //escribimos en la capa con id="mensaje" el error que el usuario tiene caracteres permitidos
        $respuesta->addAssign("mensaje","innerHTML","El nombre de usuario tiene caracteres no permitidos");
        //Cambiamos a rojo el color del texto de la capa mensaje
        $respuesta->addAssign("mensaje","style.color","red");
    }elseif (!comprobar_repetidos($entrada)){
        //escribimos en la capa con id="mensaje" el error que el usuario está repetido
        $respuesta->addAssign("mensaje","innerHTML","El nombre de usuario escrito ya está en uso");
        //Cambiamos a rojo el color del texto de la capa mensaje
        $respuesta->addAssign("mensaje","style.color","red");
    }else{
        //es que todo ha ido bien
        //escribimos en la capa con id="mensaje" que todo ha ido bien
        $respuesta->addAssign("mensaje","innerHTML","Todo correcto");
        //Cambiamos a azul el color del texto de la capa mensaje
        $respuesta->addAssign("mensaje","style.color","blue");
    }

    //tenemos que devolver la instanciación del objeto xajaxResponse
    return $respuesta;
}
```

Como podemos ver, primero creamos una instancia de `xajaxResponse`, para enviar respuestas Ajax a la página.

Luego realiza varias comprobaciones de cosas que podrían fallar en un nombre de usuario, comenzando con la verificación de que el usuario no sea la cadena vacía. En cada comprobación que ha ido mal se hacen dos cosas:

Primero se muestra un mensaje de error en la capa que tiene como identificador `"mensaje"`.

```
$respuesta->addAssign("mensaje","innerHTML","Debes escribir algo como nombre de usuario");
```

Luego, se cambia el color de la capa a rojo para que el error se visualice en rojo y sea fácil de identificar. Esta es una utilidad interesante de xajax que todavía no habíamos visto anteriormente.

```
$respuesta->addAssign("mensaje","style.color","red");
```

Como podemos entender por la anterior instrucción, en realidad con xajax podemos acceder y modificar cualquier atributo de estilo de una capa desde PHP. Por ejemplo, para cambiar el color de fondo de una capa podríamos escribir esto:

```
$respuesta->addAssign("mensaje","style.background","black");
```

Si el usuario era válido simplemente mostramos otro mensaje, esta vez con un "Todo correcto" y actualizamos el color del texto a azul.

Con esto ya tenemos hecho todo lo que habría que ver de este ejemplo, que no era nada difícil. Podemos ver el código fuente completo aquí:

```
<?
//incluimos la clase ajax
require ('xajax/xajax.inc.php');

//instanciamos el objeto de la clase xajax
$xajax = new xajax();
$xajax->setCharEncoding('ISO-8859-1');
$xajax->decodeUTF8InputOn();

function procesar_formulario($form_entrada){
    //creo el xajaxResponse para generar una salida
    $respuesta = new xajaxResponse('ISO-8859-1');

    //validación
    $error_form = "";
    if ($form_entrada["nombre"] == "")
        $error_form = "Debes escribir tu nombre";
    elseif ($form_entrada["apellidos"] == "")
        $error_form = "Debes escribir tus apellidos";
    elseif (!isset($form_entrada["acepto"]))
        $error_form = "Debes aceptar los términos y condiciones";

    //compruebo resultado de la validación
    if ($error_form != ""){
        //Hubo un error en el formulario
        //en la capa donde se muestran mensajes, muestro el error
        $respuesta->addAssign("mensaje","innerHTML","<span
style='color:red;'>$error_form</span>");
    }else{
        //es que no hubo error en el formulario
        $salida = "Hemos procesado esto:<p>";
        $salida .= "Nombre: " . $form_entrada["nombre"];
        $salida .= "<br>Apellidos: " . $form_entrada["apellidos"];

        //mostramos en capa mensaje el texto que está todo correcto
        $respuesta->addAssign("mensaje","innerHTML","<span
style='color:blue;'>Todo
correcto... Muchas gracias!</span>");
        //escribimos en la capa con id="capaformulario" el texto que aparece en $salida
        $respuesta->addAssign("capaformulario","innerHTML",$salida);

        //tenemos que devolver la instanciación del objeto xajaxResponse
    }
    return $respuesta;
}

//registramos la función creada anteriormente al objeto xajax
$xajax->registerFunction("procesar_formulario");

//El objeto xajax tiene que procesar cualquier petición
$xajax->processRequests();
?>
<html>
<head>

    <title>Enviar y procesar un formulario con Ajax y PHP</title>
    <?
    //En el <head> indicamos al objeto xajax se encargue de generar el javascript
    necesario
    $xajax->printJavascript("xajax/");
    ?>
```

```

</head>

<body>
<h1>Recibir y procesar formulario con Ajax y PHP</h1>
<div id="mensaje">
Rellena los datos de este formulario y pulsa "Enviar"
</div>
<br />
<div id="capaformulario">
<form id="formulario">
Nombre: <input type="text" name="nombre" />
<br />
Apellidos: <input type="text" name="apellidos" />
<br />
<input type="checkbox" name="acepto" value="1" /> Acepto los términos y condiciones ;)
<br />
<input
                                type="button"                                value="Enviar"
onclick="xajax_procesar_formulario(xajax.getFormValues('formulario'))" />
</form>
</div>

</body>
</html>

```

Mostrar un formulario al pulsar un enlace

La idea es que todo el proceso de comentar el artículo permanezca en la misma página, de modo que el usuario no tenga que recargar la página para comentar el artículo.

En un principio en la página no aparece el formulario para comentar el artículo, sino un enlace. Al pulsar el enlace se muestra el formulario en la página (siempre sin recargar el contenido de la página entera). Luego se envía el formulario y se procesa, también en la misma página.

Para explicar este ejercicio tenemos que basarnos en otro ejercicio anterior, en el que [enviábamos y procesábamos un formulario con Ajax y PHP](#). Lo nuevo en este artículo es la parte de mostrar un formulario en la misma página al pulsar un enlace.

Vamos a tener este HTML, donde inicialmente no está el formulario, pero tenemos el enlace para mostrarlo.

```

<div id="mensaje"></div>
<br />
<div id="capaformulario">
<a href="#" onclick="xajax muestra formulario()">Escribe un comentario del artículo</a>.
</div>

```

El enlace llama a una función PHP que se ejecutará por medio de Ajax:

```

function muestra formulario(){
    //creo el xajaxResponse para generar una salida
    $respuesta = new xajaxResponse('ISO-8859-1');

    $codigo_formul = '<form id="formulario">
Nombre: <input type="text" name="nombre" />
<br />
Apellidos: <input type="text" name="apellidos" />
<br />
<textarea name="cuerpo"></textarea>
<br />
<input
                                type="button"                                value="Enviar"
onclick="xajax_procesar_formulario(xajax.getFormValues(\'formulario\'))" />
</form>
';

    //mostramos en capa capaformulario el texto código HTML del formulario
    $respuesta->addAssign("capaformulario","innerHTML",$codigo formul);
    return $respuesta;
}

```

```
}

```

En esta función se crea en una variable cadena, donde se introduce el código del formulario. Luego, con el método `addAssign()` del objeto clase `xajaxResponse` se actualiza la capa `"capaformulario"`, para mostrar en la página el formulario.

El resto del ejercicio es exactamente el mismo que para el artículo [enviar y procesar un formulario con Ajax y PHP](#).

Podemos ver el código completo a continuación.

```
<?
//incluimos la clase ajax
require ('xajax/xajax.inc.php');

//instanciamos el objeto de la clase xajax
$xajax = new xajax();
$xajax->setCharEncoding('ISO-8859-1');
$xajax->decodeUTF8InputOn();

function muestra_formulario(){
    //creo el xajaxResponse para generar una salida
    $respuesta = new xajaxResponse('ISO-8859-1');

    $codigo_formul = '<form id="formulario">
Nombre: <input type="text" name="nombre" />
<br />
Apellidos: <input type="text" name="apellidos" />
<br />
<textarea name="cuerpo"></textarea>
<br />
<input
                                type="button"                                value="Enviar"
onclick="xajax procesar formulario(xajax.getFormValues(\'formulario\'))" />
</form>
';
    //mostramos en capa capaformulario el texto código HTML del formulario
    $respuesta->addAssign("capaformulario","innerHTML",$codigo_formul);
    return $respuesta;
}

function procesar_formulario($form_entrada){
    //creo el xajaxResponse para generar una salida
    $respuesta = new xajaxResponse('ISO-8859-1');

    //validación
    $error_form = "";
    if ($form_entrada["nombre"] == "")
        $error_form = "Debes escribir tu nombre";
    elseif ($form_entrada["apellidos"] == "")
        $error_form = "Debes escribir tus apellidos";
    elseif (strlen($form_entrada["cuerpo"]) < 10)
        $error_form = "El comentario debe tener al menos 10 caracteres";

    //compruebo resultado de la validación
    if ($error_form != ""){
        //Hubo un error en el formulario
        //en la capa donde se muestran mensajes, muestro el error
        $respuesta->addAssign("mensaje","innerHTML","<span
style='color:red;'>$error_form</span>");
    }else{
        //es que no hubo error en el formulario
        $salida = "Hemos procesado esto:<p>";
        $salida .= "Nombre: " . $form_entrada["nombre"];
        $salida .= "<br>Apellidos: " . $form_entrada["apellidos"];
        $salida .= "<br>Comentario: " . $form_entrada["cuerpo"];

        //mostramos en capa mensaje el texto que está todo correcto
        $respuesta->addAssign("mensaje","innerHTML","<span
                                style='color:blue;'>Todo
correcto... Muchas gracias!</span>");
    }
}
```

```

        //escribimos en la capa con id="capaformulario" el texto que aparece en $salida
        $respuesta->addAssign("capaformulario","innerHTML",$salida);

        //tenemos que devolver la instanciación del objeto xajaxResponse
    }
    return $respuesta;
}
//registramos la función creada anteriormente al objeto xajax
$xajax->registerFunction("muestra_formulario");
$xajax->registerFunction("procesar_formulario");

//El objeto xajax tiene que procesar cualquier petición
$xajax->processRequests();
?>
<html>
<head>

    <title>Enviar y procesar un formulario con Ajax y PHP</title>
    <?
    //En el <head> indicamos al objeto xajax se encargue de generar el javascript
    necesario
    $xajax->printJavascript("xajax/");
    ?>
</head>

<body>
<h1>Recibir y procesar formulario con Ajax y PHP</h1>
Esto se supone que es un artículo, con muchas informaciones de lo que sea. Tiene
principalmente texto.
<p>
En este ejemplo vamos a hacer un sistema para que se puedan realizar comentarios al
artículo, pero estos comentarios los vamos a recoger con Ajax.
<p>
Utilizaremos Ajax para mostrar el formulario de contacto para escribir el formulario y
también a la hora de recibir y procesar datos del formulario de alta del comentario.

<div id="mensaje"></div>
<br />
<div id="capaformulario">
<a href="#" onclick="xajax_muestra_formulario()">Escribe un comentario del artículo</a>.
</div>

</body>
</html>

```

Manejo de sesiones

Para realizar el ejemplo hemos construido un juego del ahorcado, para adivinar palabras. En el juego tendremos que meter varias variables en la sesión, como la palabra a adivinar o los fallos que lleva el jugador.

El trabajo con sesiones en Ajax y en concreto utilizando el framework de Xajax, no difiere del que ya conocemos para aplicaciones PHP generales, en resumen:

1) Tenemos que iniciar la sesión antes de enviar ninguna información o texto al navegador.

```
session_start();
```

2) Tenemos que acceder a variables de sesión a través de \$_SESSION.

```
$_SESSION["nombre_variable"] = "valor cualquiera";
```

En cuanto al juego del ahorcado, me figuro que es de sobra conocido por todos. Consiste en adivinar una palabra, probando letra a letra. Si la letra está en la palabra, se descubre y si la letra no estaba en la palabra se apunta un fallo. Se gana cuando se han adivinado todas las letras de la palabra y se pierde si se llega al máximo de los fallos permitidos.

En Xajax podremos utilizar variables de sesión de manera normal. Tenemos que asegurarnos que la sesión se abra, por lo que vamos a iniciar el código con la apertura de la sesión. Luego meteremos el `include` con la librería xajax y la instanciación del objeto.

```
<?
session start();

//incluimos la clase ajax
require ('xajax/xajax.inc.php');

//instanciamos el objeto de la clase xajax
$xajax = new xajax();
$xajax->setCharEncoding('ISO-8859-1');
$xajax->decodeUTF8InputOn();
```

Luego tendremos que crear una serie de funciones PHP, algunas para llamar como respuesta a acciones del usuario por medio de Ajax. Tenemos dos funciones principales:

Función para iniciar el juego.

- ✓ Esta función la llamaremos al terminar de cargar la página y cuando el usuario pulse el botón "Reiniciar".
- ✓ Elige una palabra aleatoriamente, inicializa los fallos, los aciertos, etc.
- ✓ Todas esas variables que se inicializan se tienen que conservar durante todo el juego, por lo que se guardan en variables de sesión.

```
function iniciar(){
    $palabras = array("murciélago", "otorrinolaringologo", "constitución",
    "deshidratación", "laboratorio", "acomodarse", "microperforado", "descontrolados",
    "superproducción");
    //defino un número aleatorio para sacar una palabra entre las posibles
    mt_srand(time());
    $indice_aleatorio = mt_rand(0, count($palabras)-1);

    //creo variable de sesión con la palabra
    $SESSION["palabra"] = $palabras[$indice_aleatorio];
    //creo variable de sesión con los aciertos
    $SESSION["aciertos"] = array();
    //creo una variable con el número de fallos
    $SESSION["fallos"] = 0;
    //creo la variable para decir que no ha perdido
    $SESSION["sin jugar"] = false;

    //genero lo que se verá de la palabra
    $palabra_vista = genera_palabra_aciertos();
    $fallos_actuales = genera_fallos_actuales();

    $respuesta = new xajaxResponse('ISO-8859-1');
    $respuesta->addAssign("palabra", "innerHTML", $palabra_vista);
    $respuesta->addAssign("fallos", "innerHTML", $fallos_actuales);
    return $respuesta;
}
```

Hay que fijarse en el momento que se crean las variables de sesión, con el array asociativo `$SESSION["indice de variable"]`, tal como estamos acostumbrados a hacer habitualmente.

Luego nos fijamos en que se genera la palabra con los aciertos y los fallos actuales y se meten por ajax como contenidos de las capas, con el método `addAssign()` de xajax.

```
$respuesta->addAssign("palabra", "innerHTML", $palabra_vista);
$respuesta->addAssign("fallos", "innerHTML", $fallos_actuales);
```

Función para probar una letra y ver si está en la palabra buscada.

Esta función la llamaremos desde Javascript con Ajax cuando el usuario escriba una letra y pulse el botón para probarla.

Veremos que hace uso de las variables de sesión creadas en la función anterior, de la manera habitual que venimos trabajando, con el array asociativo `$ _SESSION`.

```
function probar_letra($letra){
    global $fallos_permitidos;

    $respuesta = new xajaxResponse('ISO-8859-1');

    //compruebo que he recibido una letra
    if($_SESSION["sin_jugar"]){
        $advierte_perdido = "<span style='color:red'>Ya has terminado!</span><br>" .
        genera_fallos_actuales() . "";
        $respuesta->addAssign("fallos","innerHTML",$advierte_perdido);
    }elseif (strlen($letra)!=1){
        $envia_letra = "<span style='color:red'>No he recibido una letra!</span><br>" .
        genera_fallos_actuales();
        $respuesta->addAssign("fallos","innerHTML",$envia_letra);
    }else{
        //miro si la letra está entre las de la palabra
        if (strpos($_SESSION["palabra"],$letra) === false){
            //es que no estaba
            $_SESSION["fallos"] ++;
            //actualizo los fallos
            $fallos_actuales = genera_fallos_actuales();
            $respuesta->addAssign("fallos","innerHTML",$fallos_actuales);
            //compruebo que no me haya pasado de fallos
            if ($_SESSION["fallos"] == $fallos_permitidos){
                //si ha perdido, le muestro la palabra
                $palabra_entera = "Has perdido! la palabra era: <b>" .
                genera_palabra_entera() . "</b>";
                $respuesta->addAssign("palabra","innerHTML",$palabra_entera);
                $_SESSION["sin_jugar"] = true;
            }
        }else{
            //es que la letra está en la cadena
            $_SESSION["aciertos"][count($_SESSION["aciertos"])] = $letra;
            //genero lo que se verá de la palabra
            $palabra_vista = genera_palabra_aciertos();
            //compruebo si has ganado ya
            if (strpos($palabra_vista,"_")==false){
                //si no hay un " " en la palabra vista es que ya ha visto todas las letras
                $palabra_vista = "Has ganado! la palabra es: <b>" . $palabra_vista . "</b>";
                $_SESSION["sin_jugar"] = true;
            }
            $respuesta->addAssign("palabra","innerHTML",$palabra_vista);
        }
    }
    return $respuesta;
}
```

La función es un poco larga, porque implementa casi toda la lógica del juego. Ahora no vamos a dar explicaciones sobre la lógica del juego del ahorcado, pero podemos ver que la función está comentada, así se pueda entender qué se hace en cada paso.

Comprobaremos que se van accediendo a las variables de sesión y que se van generando respuestas a través de los métodos del objeto `xajaxResponse`.

Conclusión sobre variables de sesión en Ajax

Es indiferente que estemos trabajando dentro de Ajax por lo que respecta al manejo de la sesión con PHP. Las variables de sesión almacenan sus valores y los recuerdan sin problemas

durante toda la ejecución de la aplicación, en las distintas llamadas generadas a PHP a través de Ajax

El juego del ahorcado tiene cierta complejidad, en la que no hemos querido ahondar demasiado. Todavía habría que hacer cosas importantes para que fuera totalmente funcional, como la gestión de acentos para asegurarse que, cuando el usuario pruebe si hay una vocal, nos muestre la vocal tanto si está acentuada como si no. En el juego actual si probamos, por ejemplo la "e" no nos mostraría la "é" (e con tilde), si es que la hay.

Veremos aquí el código completo del juego, que nos puede dar más pistas sobre el trabajo con PHP y Xajax.

```
<?
session_start();

//incluimos la clase ajax
require ('xajax/xajax.inc.php');

//instanciamos el objeto de la clase xajax
$xajax = new xajax();
$xajax->setCharEncoding('ISO-8859-1');
$xajax->decodeUTF8InputOn();

$fallos_permitidos=5;

function iniciar(){
    $palabras = array("murciélago", "otorrinolaringologo", "constitución",
    "deshidratación", "laboratorio", "acomodarse", "microperforado", "descontrolados",
    "superproducción");
    //defino un número aleatorio para sacar una palabra entre las posibles
    mt_srand(time());
    $indice_aleatorio = mt_rand(0,count($palabras)-1);

    //creo variable de sesión con la palabra
    $_SESSION["palabra"] = $palabras[$indice_aleatorio];
    //creo variable de sesión con los aciertos
    $_SESSION["aciertos"] = array();
    //creo una variable con el número de fallos
    $_SESSION["fallos"] = 0;
    //creo la variable para decir que no ha perdido
    $_SESSION["sin_jugar"] = false;

    //genero lo que se verá de la palabra
    $palabra_vista = genera_palabra_aciertos();
    $fallos_actuales = genera_fallos_actuales();

    $respuesta = new xajaxResponse('ISO-8859-1');
    $respuesta->addAssign("palabra","innerHTML",$palabra_vista);
    $respuesta->addAssign("fallos","innerHTML",$fallos_actuales);
    return $respuesta;
}

function genera_palabra_aciertos(){
    $cadena_palabra="";
    //para cada una de las letras de la palabra a buscar
    for ($i=0; $i<strlen($_SESSION["palabra"]); $i++){
        //si la letra está en los aciertos, la muestro
        if (in_array(substr($_SESSION["palabra"],$i,1), $_SESSION["aciertos"])){
            //si la letra actual está en el array de aciertos, la muestro
            $cadena_palabra .= " " . substr($_SESSION["palabra"],$i,1) . " ";
        }else{
            //si la letra no está en el array de aciertos, no la muestro
            $cadena_palabra .= " _ ";
        }
    }
    return $cadena_palabra;
}

function genera_palabra_entera(){
    $cadena_palabra="";
    //para cada una de las letras de la palabra a buscar
```

```

    for ($i=0; $i<strlen($_SESSION["palabra"]); $i++){
        $cadena_palabra .= " " . substr($_SESSION["palabra"],$i,1) . " ";
    }
    return $cadena_palabra;
}

function genera_fallos_actuales(){
    global $fallos_permitidos;
    $factuales = "Fallos: " . $_SESSION["fallos"] . "/" . $fallos_permitidos;
    return $factuales;
}

function probar_letra($letra){
    global $fallos_permitidos;

    $respuesta = new xajaxResponse('ISO-8859-1');

    //compruebo que he recibido una letra
    if($_SESSION["sin_jugar"]){
        $advierte_perdido = "<span style='color:red'>Ya has terminado!</span><br>" .
        genera_fallos_actuales() . " ";
        $respuesta->addAssign("fallos","innerHTML",$advierte_perdido);
    }elseif (strlen($letra)!=1){
        $envia_letra = "<span style='color:red'>No he recibido una letra!</span><br>" .
        genera_fallos_actuales();
        $respuesta->addAssign("fallos","innerHTML",$envia_letra);
    }else{
        //miro si la letra está entre las de la palabra
        if (strpos($_SESSION["palabra"],$letra) === false){
            //es que no estaba
            $_SESSION["fallos"] ++;
            //actualizo los fallos
            $fallos_actuales = genera_fallos_actuales();
            $respuesta->addAssign("fallos","innerHTML",$fallos_actuales);
            //compruebo que no me haya pasado de fallos
            if ($_SESSION["fallos"] == $fallos_permitidos){
                //si ha perdido, le muestro la palabra
                $palabra_entera = "Has perdido! la palabra era: <b>" .
                genera_palabra_entera() . "</b>";
                $respuesta->addAssign("palabra","innerHTML",$palabra_entera);
                $_SESSION["sin_jugar"] = true;
            }
        }else{
            //es que la letra está en la cadena
            $_SESSION["aciertos"][count($_SESSION["aciertos"])] = $letra;
            //genero lo que se verá de la palabra
            $palabra_vista = genera_palabra_aciertos();
            //compruebo si has ganado ya
            if (strpos($palabra_vista," ")===false){
                //si no hay un " " en la palabra vista es que ya ha visto todas las letras
                $palabra_vista = "Has ganado! la palabra es: <b>" . $palabra_vista . "</b>";
                $_SESSION["sin_jugar"] = true;
            }
            $respuesta->addAssign("palabra","innerHTML",$palabra_vista);
        }
    }
    return $respuesta;
}

//registramos funciones
$xajax->registerFunction("iniciar");
$xajax->registerFunction("probar_letra");

//El objeto xajax tiene que procesar cualquier petición
$xajax->processRequests();
?>
<html>
<head>
    <title>Ahorcado Ajax</title>
    <?
    //En el <head> indicamos al objeto xajax se encargue de generar el javascript
    necesario
    $xajax->printJavascript("xajax/");
    ?>
</head>

<body onload="xajax_iniciar()">

```

```

<h1>Ahorcado en Ajax</h1>
<div id="palabra">
</div>
<br>
<div id="formulario">
<form id="formulario" onsubmit="return false;">
Escribe una letra: <input type="text" name="letra" size=2 maxlength="1">
<br>
<input
                        type="button"                        value="Probar"                        letra"
onclick="xajax probar letra(this.form.letra.value);this.form.letra.value=''">
</form>
</div>
<br>
<div id="fallos">
</div>
<br>

<form id="reiniciar" onsubmit="return false;">
<input type="button" value="reiniciar" onclick="xajax iniciar()">
</form>

</body>
</html>

```

Comprobar la validez de una URL

En este ejemplo tendremos un formulario donde el usuario podrá escribir la URL de un sitio web. Con Ajax comprobaremos su validez, desde el punto de vista sintáctico (tiene que empezar por `http://`) y luego haremos una pequeña utilidad para que el visitante pueda ver el sitio web que se proporciona dentro de un `iframe`. Si el sitio web se ve correctamente dentro del `iframe` es que la URL es correcta.

De nuevo comento que este ejercicio lo hemos creado ayudándonos de las librerías Xajax, por lo que no explicaré los detalles del uso de Xajax.

Vamos a empezar viendo el formulario HTML que hemos creado:

```

<form name="formulario">
  URL: <input type="text" name="url">
  <input
        type="button"
        name="comprobar url"
        value="Comprobar
URL"
onclick="xajax comprobar url(document.formulario.url.value)">
  <div id="comprobacion"></div>
  <br>
  Título: <input type="text" name="titulo">
</form>

```

Simplemente tiene un campo de texto de la URL y un botón, que al pulsarlo se lanza la comprobación. Luego encontramos un `<div id="comprobacion">`, donde mostraremos el resultado de la validación por medio de Ajax. El último campo de texto no lo utilizamos para nada.

Al pulsar el botón, evento `onclick`, se llama a una función que hace uso de Ajax. Esta función está escrita en PHP. Mediante xajax conseguimos que se pueda ejecutar con una llamada desde Javascript a esta función PHP.

```

function comprobar_url($miurl){
  if($miurl == ""){
    $texto_result = "Debes escribir una URL";
  }elseif(substr($miurl, 0, 7) != "http://"){
    $texto_result = "La URL debe comenzar por http://";
  }else{
    $texto result = '<iframe width="600" height="200" src="' . $miurl . '"></iframe>
<br>

```

```

        <a href="#" onclick="xajax_NO_comprobar_url()">Ocultar...</a>;
    }
    $respuesta = new xajaxResponse('ISO-8859-1');
    $respuesta->addAssign("comprobacion","innerHTML",$texto result);
    return $respuesta;
}

```

La función recibe en el parámetro `$miurl` la URL que se desea validar. Primero realiza un par de validaciones simples, que generan mensajes de error si no la URL es un `string` vacío o si la URL no comienza por `"http://"`.

Luego, para que el usuario valide visualmente que la dirección que ha escrito corresponde con la URL que quería enviar, se muestra un `<iframe>` con la URL de la página que ha escrito en el formulario. Si la URL estaba bien escrita el usuario podrá verla dentro del `<iframe>`. Además, para comodidad del usuario, hemos puesto debajo un enlace para ocultar el `<iframe>`. Ese enlace llama a otra función por medio de Ajax que borra de la página el `<iframe>`.

```

function NO_comprobar_url(){
    $texto result = '';
    $respuesta = new xajaxResponse('ISO-8859-1');
    $respuesta->addAssign("comprobacion","innerHTML",$texto_result);
    return $respuesta;
}

```

Dejamos el código fuente completo:

```

<?
session_start();

//incluimos la clase ajax
require ('xajax/xajax.inc.php');

//instanciamos el objeto de la clase xajax
$xajax = new xajax();
$xajax->setCharEncoding('ISO-8859-1');
$xajax->decodeUTF8InputOn();

function comprobar url($miurl){
    if($miurl == ""){
        $texto_result = "Debes escribir una URL";
    }elseif(substr($miurl, 0, 7) != "http://"){
        $texto_result = "La URL debe comenzar por http://";
    }else{
        $texto result = '<iframe width="600" height="200" src="' . $miurl . '"></iframe>
        <br>
        <a href="#" onclick="xajax_NO_comprobar_url()">Ocultar...</a>;
    }
    $respuesta = new xajaxResponse('ISO-8859-1');
    $respuesta->addAssign("comprobacion","innerHTML",$texto result);
    return $respuesta;
}

function NO_comprobar_url(){
    $texto_result = '';
    $respuesta = new xajaxResponse('ISO-8859-1');
    $respuesta->addAssign("comprobacion","innerHTML",$texto result);
    return $respuesta;
}

$xajax->registerFunction("comprobar url");
$xajax->registerFunction("NO comprobar url");

//El objeto xajax tiene que procesar cualquier petición
$xajax->processRequests();
?>
<html>
<head>
    <title>Comprobar URL por iframe y ajax</title>
</head>

```

```
//En el <head> indicamos al objeto xajax se encargue de generar el javascript
necesario
$xajax->printJavascript("xajax/");
?>

</head>

<body>

<form name="formulario">
URL: <input type="text" name="url">
<input type="button" name="comprobar_url" value="Comprobar URL"
onclick="xajax_comprobar_url(document.formulario.url.value)">
<div id="comprobacion"></div>
<br>
Título: <input type="text" name="titulo">
</form>

</body>
</html>
```

Validación previa de un campo de texto

Este componente de formulario, que incorpora Javascript + Ajax + PHP + Base de datos MySQL, sirve para que un visitante que escribe un nombre de país, lo valide apretando un botón para ver si existe o se ha escrito correctamente. En caso que el usuario escriba sólo unas letras del nombre del país, por ejemplo "es", cuando aprieta el enlace de validar, el componente busca todos los países que tienen ese texto y los muestra para que el usuario elija el que desea.

Hay temas importantes que no vamos a explicar porque ya se vieron en capítulos anteriores.

El formulario HTML

Comenzamos mostrando el formulario que hemos utilizado para construir el ejemplo:

```
<div id="todo_formulario">
  <form action="#" method="post" name="f1" id="f1">
    <div id="input_pais">
      Nombre país: <input type="text" name="nombre_pais"> <a style='cursor:pointer;text-decoration:underline; color:#0000ff;'
onclick="xajax_validar_pais(document.f1.nombre_pais.value);">Validar país</a>
    </div>
    <div id="validador_pais"></div>
    <p>
      Año de visita: <input type="text" name="ano" size="8">
    </p>
    <input type="button" onclick="xajax_procesar_formulario(xajax.getFormValues('f1'))"
value="Enviar">
  </form>
</div>
```

Tenemos un campo de texto donde el usuario escribe el país, un enlace al lado para validarlo y unas capas o etiquetas `<div>` que delimitan ciertas áreas donde vamos a mostrar datos desde Ajax. Luego hay un botón para enviar el formulario, que también se procesa con una llamada a una función Javascript que hace uso de Ajax.

La llamada Ajax para validar el país

Tenemos una función creada con PHP, que hace uso de XajaX para poder procesarla con Ajax, que es la que se encarga de validar lo que haya escrito el usuario en el campo país.

```
xajax_validar_pais(document.f1.nombre_pais.value);
```

La función recibe el `value` del campo `input nombre_pais`, es decir, lo que haya escrito en el momento de pulsar el enlace.

```
function validar_pais($nombre_pais){
```

```

if (strlen($nombre_pais)<2) {
    $campo_validacion = "Debes escribir al menos dos letras del campo país.";
}else{
    $ssql = "select * from pais where nombre pais like '%$nombre_pais%'";
    $rs = mysql_query($ssql);
    if (mysql_num_rows($rs)==0){
        $campo_validacion = "No he encontrado países con ese nombre";
    }else{
        $campo_validacion = "<i>Se encontraron " . mysql_num_rows($rs) . " posibles países.</i>";
        $campo_validacion .= '<div style="margin:3px;">';
        while ($fila = mysql_fetch_object($rs)){
            $campo_validacion .= "<a style='cursor:pointer;text-decoration:underline; color:#0000ff;' onclick='selecciona_pais(\"" . $fila->nombre_pais . "\");'>";
            $campo_validacion .= $fila->nombre_pais . "</a><br>";
        }
        $campo_validacion .= '</div>';
    }
    $campo_validacion = '<div style="border: 2px solid #0000cc; font-size: 8pt; padding:5px; margin-top:10px; width: 300px;">' . $campo_validacion . '</div>';

    $respuesta = new xajaxResponse('ISO-8859-1');
    $respuesta->addAssign("validador_pais","innerHTML",$campo_validacion);
    return $respuesta;
}

```

Lo primero que se comprueba es que el usuario haya escrito al menos dos caracteres en el campo país.

Luego se hace una consulta en la base de datos para seleccionar todos los países que tienen el texto escrito en el campo. Si no encuentra ninguno se muestra un mensaje de error. Si encuentra uno o más países, se muestra el listado de naciones encontradas para que el usuario seleccione cuál es la que le interesa. Se utilizan las funciones proporcionadas por un objeto del framework xajax (de la clase `xajaxResponse`) para mostrar los resultados en la página si refrescar el documento.

Función para procesar el formulario con Ajax y PHP

Para procesar el formulario utilizamos también Ajax y PHP. Para ello hemos realizado la siguiente función PHP que se ejecuta por medio de Ajax:

```

function procesar_formulario($formulario){
    $respuesta = new xajaxResponse('ISO-8859-1');
    if (!isset($formulario["nombre_validado"])){
        $respuesta->addAssign("validador_pais","innerHTML",<div style="border: 2px solid #cc0000; font-size: 8pt; padding:5px; margin-top:10px; width: 300px;">Tienes que validar el país.</div>');
    }else{
        $respuesta->addAssign("todo_formulario","innerHTML","Todo correcto!");
    }
    return $respuesta;
}

```

El formulario comprueba que se haya validado anteriormente el país, para que permita procesarlo.

Javascript para escoger el país deseado

Luego hay una función que también se debe comentar, que es una función Javascript pura, que se procesa totalmente en el cliente, cuando el usuario selecciona el país que desea de los países encontrados al validar el país.

```

<script>
function selecciona_pais(nombre){
    document.f1.nombre_pais.value=nombre
    document.getElementById("validador_pais").innerHTML='<div style="border: 2px solid #00cc00; font-size: 8pt; padding:5px; margin-top:10px; width: 300px;">Validado correctamente</div>';
}

```

```

document.getElementById("input_pais").innerHTML='<input
name="nombre_validado" value="' + nombre + '">' + nombre
type="hidden"
}'
</script>

```

Esta función se ejecuta al pulsar cualquier país de la lista de países encontrados y actualiza el contenido HTML de un par de capas, para mostrar el resultado en la interfaz de usuario y generar un campo hidden con el nombre del país ya validado.

El código completo lo mostramos a continuación:

```

<?
//incluimos la clase ajax
require ('xajax/xajax.inc.php');

//instanciamos el objeto de la clase xajax
$xajax = new xajax();
$xajax->setCharEncoding('ISO-8859-1');
$xajax->decodeUTF8InputOn();

//conecto con la base de datos
$connectid = mysql_connect("localhost", "root", "");
mysql_select_db("xajax",$connectid);

function validar_pais($nombre_pais){
    if (strlen($nombre_pais)<2) {
        $campo_validacion = "Debes escribir al menos dos letras del campo país.";
    }else{
        $ssql = "select * from pais where nombre_pais like '%$nombre_pais%'";
        $rs = mysql_query($ssql);
        if (mysql_num_rows($rs)==0){
            $campo_validacion = "No he encontrado países con ese nombre";
        }else{
            $campo_validacion = "<i>Se encontraron " . mysql_num_rows($rs) . " posibles
países.</i>";
            $campo_validacion .= '<div style="margin:3px;">';
            while ($fila = mysql_fetch_object($rs)){
                $campo_validacion .= "<a style='cursor:pointer;text-decoration:underline;
color:#0000ff;' onclick='selecciona_pais(\"" . $fila->nombre_pais . "\");'>";
                $campo_validacion .= $fila->nombre_pais . '</a><br>';
            }
            $campo_validacion .= '</div>';
        }
    }
    $campo_validacion = '<div style="border: 2px solid #0000cc; font-size: 8pt;
padding:5px; margin-top:10px; width: 300px;">' . $campo_validacion . '</div>';

    $respuesta = new xajaxResponse('ISO-8859-1');
    $respuesta->addAssign("validador_pais","innerHTML",$campo_validacion);
    return $respuesta;
}

function procesar_formulario($formulario){
    $respuesta = new xajaxResponse('ISO-8859-1');
    if (!isset($formulario["nombre_validado"])){
        $respuesta->addAssign("validador_pais","innerHTML",<div style="border: 2px solid
#cc0000; font-size: 8pt; padding:5px; margin-top:10px; width: 300px;">Tienes que validar
el país.</div>');
    }else{
        $respuesta->addAssign("todo_formulario","innerHTML",'Todo correcto!');
    }
    return $respuesta;
}

//registramos funciones
$xajax->registerFunction("validar_pais");
$xajax->registerFunction("procesar_formulario");

//El objeto xajax tiene que procesar cualquier petición
$xajax->processRequests();
?>
<html>
<head>
<title>Validador campo formulario online</title>

```

```

<?
//En el <head> indicamos al objeto xajax se encargue de generar el javascript
necesario
$xajax->printJavascript("xajax/");
?>
<script>
function selecciona_pais(nombre){
    document.fl.nombre_pais.value=nombre
    document.getElementById("validador_pais").innerHTML='<div style="border: 2px solid
#00cc00; font-size: 8pt; padding:5px; margin-top:10px; width: 300px;">Validado
correctamente</div>';
    document.getElementById("input_pais").innerHTML='<input
name="nombre validado" value="" + nombre + ">' + nombre
    }
}
</script>
</head>

<body style="font-family: arial, verdana;">
<div id="todo formulario">
    <form action="#" method="post" name="f1" id="f1">
        <div id="input pais">
            Nombre país: <input type="text" name="nombre pais"> <a style='cursor:pointer;text-
decoration:underline; color:#0000ff;'
onclick="xajax_validar_pais(document.fl.nombre_pais.value);">Validar pais</a>
        </div>
        <div id="validador pais"></div>
        <p>
            Año de visita: <input type="text" name="ano" size="8">
        </p>
        <input type="button" onclick="xajax_procesar_formulario(xajax.getFormValues('f1'))"
value="Enviar">
        </form>
    </div>

<?
mysql_close($conn);
?>
</body>
</html>

```

Script para detección de soporte a Ajax, Cookies y ActiveX

El sitio de Xajax Project ha publicado unos scripts interesantes para poder detectar si un navegador es compatible con la tecnología Ajax, para estar seguros que la web que estamos desarrollando se va a poder mostrar correctamente en cualquier cliente web que tenga el usuario. Además estos scripts sirven para mostrar mensajes de error si el navegador no tiene soporte a Ajax, de modo que el usuario sea consciente que no va a poder ver esa web convenientemente.

Estos scripts detectan las capacidades del navegador y se pueden ejecutar para mostrar mensajes de alerta si no están disponibles ciertas funcionalidades, ya sea porque el navegador del usuario no las soporta o porque estén deshabilitadas.

El script contiene tres funciones:

browserSupportsCookies()

Detecta si el navegador soporta cookies y devuelve true en caso que estén soportadas y false si no es así.

browserSupportsAjax()

Comprueba si el navegador tiene compatibilidad con la tecnología Ajax, devuelve true si es así y false si no soporta Ajax por cualquier cuestión.

ActiveXEnabledOrUnnecessary()

Esta función detecta si el navegador soporta ActiveX o bien si ActiveX es innecesario para la ejecución de Ajax. En el navegador Internet Explorer 6 Ajax se ejecuta a través de ActiveX, así que necesita disponer ActiveX para que todo funcione. Así que esta función devolverá false sólo si el navegador es Internet Explorer 6 y tiene inhabilitado ActiveX.

Las funciones no las voy a escribir en el texto de este artículo, simplemente voy a poner un link al lugar donde se muestran las funciones en la página de Xajax Project:

[http://xajax-project.org/wiki/Xajax %28any%29: Tips and Tricks: Detecting Support](http://xajax-project.org/wiki/Xajax_%28any%29:_Tips_and_Tricks:_Detecting_Support)

Qué lástima, parece que ese link ya no funciona. Pero podéis encontrar esas funciones a través de un enlace a una página en DesarrolloWeb.com donde hemos implementado estos scripts, para que los podáis ver en funcionamiento en vuestros navegadores. Así mismo, podéis ver el código fuente de la página para ver la implementación de los scripts que hemos hecho en DesarrolloWeb.com y obtener el código de las funciones en caso que cambien la URL en la página de Xajax.

<http://www.desarrolloweb.com/articulos/ejemplos/comprobar-compatibilidad-ajax.html>

Interfaz de navegación por pestañas

En la interfaz habrá varias pestañas y pulsando cada una se mostrarán unos contenidos específicos de esa solapa, todos en el mismo espacio.

Es una interfaz que habremos podido ver en varias páginas web. Nosotros vamos a realizarla con PHP y Ajax, para que se soliciten los contenidos de cada pestaña al servidor y se muestren sin necesidad de recargar la página.

Maquetación de la interfaz con CSS y HTML

Para maquetar previamente el sistema de pestañas vamos a utilizar por un lado HTML y por otro CSS para definir el aspecto.

Nuestro HTML en este ejercicio quedará así:

```
<div class="pestanas">
  <ul>
    <li id="pestanas0" class="pestanainactiva">
      <a href="javascript:void(xajax_cambia_contenido(0))">Intereses</a>
    </li>
    <li id="pestanas1" class="pestanainactiva">
      <a href="javascript:void(xajax_cambia_contenido(1))">Portfolio</a>
    </li>
    <li id="pestanas2" class="pestanainactiva">
      <a href="javascript:void(xajax_cambia_contenido(2))">Contacto</a>
    </li>
  </ul>
  <div id="cuerpopestanas" class="cuerpopestanas"></div>
</div>
```

En el HTML ya podemos ver llamadas a una función Javascript `xajax_cambia_contenido()`, que en realidad es una función, llamada `cambia_contenido()`, que hemos definido con PHP y registrado con Xajax para poder invocarla desde Javascript. Estas llamadas a `xajax_cambia_contenido()` envían un parámetro, que es el número de la pestaña pulsada.

Aparte, conviene percatarse de que cada pestaña construida tiene un identificador, por ejemplo `pestanas0` o `pestanas1`, que luego vamos a necesitar, en nuestra función PHP `cambia_contenido()`, para referirnos a estas pestañas y cambiarles el aspecto al ser pulsadas.

Además, como decía, en el CSS del ejemplo vamos a tener dos clases que merece la pena que recordemos:

`li.pestanainactiva`, con los estilos CSS de las pestañas cuando no están pulsadas.

`li.pestanaseleccionada`, con los estilos CSS de las pestañas cuando sí que están pulsadas.

Otra cosa con respecto al HTML, es que en la capa con `id="cuerpopestanas"` es donde vamos a mostrar los distintos contenidos de las solapas.

Por último, os habréis fijado que en el HTML están todas las solapas inactivas y ningún contenido en el cuerpo de las pestañas. Esto es porque el contenido del cuerpo de la interfaz lo vamos a inicializar desde Javascript cuando se termine de cargar la página, para que en ese momento se conecte por ajax al servidor y se traiga los contenidos de la pestaña que queramos que aparezca seleccionada por defecto.

Código PHP de la función para cambiar la pestaña pulsada

Con PHP y ayudándonos de Xajax vamos a definir el comportamiento de pulsar una de las pestañas de la interfaz. Tenemos que hacer lo siguiente:

- ✓ Conseguir el contenido de la pestaña pulsada y colocarlo en la capa "`cuerpopestanas`".
- ✓ Cambiar la clase CSS (class de CSS) de la pestaña pulsada a aquella donde habíamos guardado el estilo de las solapas cuando están pulsadas `li.pestanaseleccionada`.
- ✓ Para las pestañas que quedan inactivas, cambiar la clase de CSS que hemos indicado en `li.pestanainactiva`.

La función es la siguiente:

```
function cambia_contenido($num_pestanas){
    //instanciamos el objeto para generar la respuesta con ajax
    $respuesta = new xajaxResponse('ISO-8859-1');

    //defino contenidos de las pestañas
    $contenido_pestanas = array(
        'Texto 0',
        'Texto solapa 2',
        'Texto de la pestana 3');
    //Pongo el texto nuevo en el cuerpo de la interfaz de pestañas
    $respuesta->addAssign("cuerpopestanas","innerHTML",$contenido_pestanas[$num_pestanas]);
    //Pongo el estilo (de una class de css) en la pestaña pulsada
    $respuesta->addAssign("pestanas" . $num_pestanas, "className", "pestanaseleccionada");
    //Pongo la class css de las pestañas sin pulsar
    for ($i=0; $i<count($contenido_pestanas); $i++){
        if ($i != $num_pestanas){
            $respuesta->addAssign("pestanas" . $i, "className", "pestanainactiva");
        }
    }
    return $respuesta;
}
```

El código está comentado para una fácil comprensión. No obstante podemos decir:

- ✓ Recibo como parámetro de la función, en `$num_pestanas`, el número de la pestaña pulsada.
- ✓ Los contenidos de las pestañas los hemos metido en un array creado en la propia función. Los contenidos son textos que me he inventado y he metido directamente en el Array, pero en realidad estos contenidos los podríamos haber generado de cualquier otra forma o seleccionarlos de una base de datos.

- ✓ Para cambiar la clase (class de CSS) de un elemento de la página tenemos que utilizar la el método `addAssign()`. El primer valor pasado a este método es el identificador del elemento al que queremos cambiar la `class`, el segundo valor "`className`" para decirle que queremos cambiar el nombre de la clase CSS y el tercer valor el nombre de la clase que queremos asociar a ese elemento.
- ✓ Al final se hace un bucle para cada una de las pestañas, menos la que ha sido pulsada, para cambiarles la clase a la `class` CSS que hemos creado para las pestañas inactivas.

Con esa función registrada mediante ajax, podemos ejecutarla como hemos visto en el HTML al principio del artículo. Para inicializar la estructura de pestañas una vez cargada la página llamaremos a esta función PHP de la siguiente manera:

```
<script>
    window.onload = xajax_cambia_contenido(0);
</script>
```

Con esto, al terminarse de cargar la página se mostrarán los contenidos de la pestaña 0.

Código completo de la interfaz de pestañas

Para acabar, aquí queda el código completo de este ejercicio, que tiene muchos otros detalles.

```
<?
//incluimos la clase ajax
require ('xajax/xajax.inc.php');

//instanciamos el objeto de la clase xajax
$xajax = new xajax();
$xajax->setCharEncoding('ISO-8859-1');
$xajax->decodeUTF8InputOn();

//función para cambiar el contenido de las pestañas
function cambia_contenido($num pestana){
    //instanciamos el objeto para generar la respuesta con ajax
    $respuesta = new xajaxResponse('ISO-8859-1');

    $contenido pestanas = array(
        'Esto es el cuerpo de las pestañas. Puede ser tan largo como desees, que le
        pondremos unas barras de desplazamiento para que no se desborde.
        <br>
        <br>
        Lo que tengo es que poner más texto para las pestañas para probar que pasa cuando
        el texto es grande.
        <br>
        <br>
        Estilos CSS para las pestañas. Ejemplo funcionando en el que podrás ver como se
        maqueta con CSS para crear un estilo para un recuadro con diversas pestañas que podrán
        mostrar distintos contenidos...
        <br>
        <br>
        Existe una pestaña seleccionada, que es la que muestra los contenidos que se van a
        ver en este momento. En el ejemplo que muestra sólo los CSS las pestañas no las hemos
        hecho pinchables.
        <br>
        <br>
        Gracias por tu atención y por darle al scroll hacia abajo, que siempre es
        interesante probarlo todo bien.
        ',
        'Quisque quam dui, cursus ut, tempus at, accumsan eu, magna. Ut commodo, tortor at
        ultrices rutrum.
        <br>
        <br>
        Nulla nulla consequat ipsum, aliquam aliquam felis arcu eget purus. Aenean tempus.
        Nam hendrerit facilisis lectus. Donec velit enim, viverra in, pharetra sed, ornare
        vitae, mauris.
        <br>
        <br>
```

```

        Praesent vestibulum euismod turpis. Aliquam turpis arcu, cursus sed, venenatis ut,
        auctor id, elit. Nunc libero. Mauris tortor. Proin eu quam sed velit convallis
        malesuada. Curabitur tempor sem ac mauris. Aliquam felis velit, bibendum sit amet,
        auctor ultricies, consequat nec, neque. Sed quis mi. Duis tincidunt odio.
        <br>
        <br>
        Etiam tincidunt pede eu elit. Pellentesque at arcu. Phasellus mi tellus, gravida
        ac, consectetur sed, adipiscing nec, enim. Integer nisi lectus, scelerisque eu, dapibus
        vel, hendrerit eu, lacus.
        ',
        'Vestibulum ac pede vitae risus pharetra vulputate. Raesent massa diam, tempor sit
        amet, porttitor non, pretium eget pede.
        <br>
        <br>
        Praesent sed ipsum. Etiam suscipit lectus vitae eros. Phasellus eget pede. Nam
        quis ipsum. Sed vitae turpis. Sed sed ipsum vel augue dignissim tempor. Maecenas
        volutpat, tellus non eleifend pellentesque, eros eros ornare nibh, id egestas nunc quam
        quis neque. Curabitur nec justo. Vestibulum consectetur sapien et erat.
        <br>
        <br>
        Etiam sit amet dui vitae elit facilisis gravida. Sed molestie rhoncus sem. Nulla
        facilisi. Suspendisse potenti.
        ');
        //coloco el texto nuevo en el cuerpo de la interfaz de pestañas
        $respuesta=
>addAssign("cuerpopestanas","innerHTML",$contenido pestanas[$num pestana]);
        //Coloco el estilo (de una class de css) en la pestaña pulsada
        $respuesta->addAssign("pestanas" . $num_pestanas, "className", "pestanaseleccionada");
        //coloco la class css de las pestañas sin pulsar
        for ($i=0; $i<count($contenido_pestanas); $i++){
            if ($i != $num_pestanas){
                $respuesta->addAssign("pestanas" . $i, "className", "pestanainactiva");
            }
        }
        return $respuesta;
    }

    //asociamos la función creada anteriormente al objeto xajax
    $xajax->registerFunction("cambia contenido");

    //El objeto xajax tiene que procesar cualquier petición
    $xajax->processRequests();
    ?>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>
<head>
    <title>Interfaz de Pestañas Ajax y PHP</title>
    <link rel="stylesheet" type="text/css" href="estilo-pestanas3.css">
    <?
        //En el <head> indicamos al objeto xajax se encargue de generar el javascript
        necesario
        $xajax->printJavascript("xajax/");
        ?>
    </head>

    <body>

        Sistema de navegación con diversas pestañas, programado con Ajax y PHP para traer los
        contenidos de cada pestaña.

        <br>
        <br>

        <div class="pestanas">
            <ul>
                <li id="pestanas0" class="pestanainactiva"><a
                href="javascript:void(xajax_cambia_contenido(0))">Intereses</a></li>
                <li id="pestanas1" class="pestanainactiva"><a
                href="javascript:void(xajax_cambia_contenido(1))">Portfolio</a></li>
                <li id="pestanas2" class="pestanainactiva"><a
                href="javascript:void(xajax_cambia_contenido(2))">Contacto</a></li>
            </ul>
            <div id="cuerpopestanas" class="cuerpopestanas">
            </div>
        </div>

```

```
<script>
window.onload = xajax cambia contenido(0);
</script>

</body>
</html>
```

Sistema de votación con Xajax

Vamos a crear un sistema para que los usuarios puedan votar artículos o si se modifica, pues cualquier cosa que quieras. La particularidad de este sistema es que lo vamos a realizar con ajax para que la página no tenga que cargar cada vez que un usuario vote.

Esta es una opción muy útil para que el visitante vote por los artículos publicados sin tener que estar esperando a que la página se cargue, además la página queda mucho más elegante y da la sensación de estar bien programada. Seguro que es una opción que habremos visto en un montón de sitios web y que hasta ahora no sabías como hacerlo.

Este artículo hace continuación de una serie de talleres y ejemplos que hemos visto anteriormente en el manual Trabajo con Ajax en PHP utilizando Xajax. Por lo que es aconsejable que antes de empezar con este taller tengáis los aspectos básicos del funcionamiento de xajax bien aprendidos.

Creación del recuadro de votación

Lo primero que vamos a crear es el recuadro donde se va a mostrar la posibilidad de votar y las votaciones realizadas.

Este recuadro es bastante sencillo y lo único que tenemos que tener en cuenta es que hay que crear una capa, la cual va a cambiar en el momento en que el usuario haga su votación.

Un ejemplo de recuadro sería el siguiente:

```
<script>
function votar_articulo(id_articulo){

    var puntos = document.votar.puntos.value;
    xajax votar(id_articulo,puntos)

}
</script>

<div style="width:192px;font-size:12pt; margin-top:3px;">
<div ><b>Participa:</b></div>
<table cellpadding="2" cellspacing="2" border="0" width="190">
<tr>
<td width="130" align=center class=fuente8>
    <div id="votar_articulo">
        <?
        /*if ($respuesta!="")
            echo " <i>$respuesta</i> <br> ";*/
        //muestra_media_votos_pagina($id_breve);
        ?>

    Puntua el artículo:
    <br>
    <form action="sistema_votacion.php" method="post" name=votar style="margin:0px;">
    <select name="puntos">
    <option value=1>1
    <option value=2>2
    <option value=3>3
    <option value=4>4
    <option value=5>5
    </select>
```

```

<input type='button' name='votar' value='Votar' onClick='votar_articulo(1) '>
</form>

</div>
</div>
</td>
</tr>
<tr><td class=fuente8 align=center><div id=actualizar></div></td></tr>
</table>

</div>

```

El único punto importante a comentar es que hemos utilizado una función javascript para llamar a la función xajax que vamos a crear después.

Una vez que tenemos creado el recuadro tenemos que irnos a la página donde vamos a colocar el recuadro y empezar a construir las funciones xajax.

Antes de empezar tenemos que tener creadas ciertas tablas en nuestra base de datos. Principalmente tendremos que tener una tabla `articulo_voto` donde se almacenará la información del artículo y del voto. Sería algo como esto:

	Campo	Tipo	Collation	Atributos	Nulo	Predeterminado	Extra
<input type="checkbox"/>	id_voto	int(5)			No		auto_increment
<input type="checkbox"/>	votacion	int(2)			No	0	
<input type="checkbox"/>	id_articulo	int(9)		UNSIGNED	No	0	

Para ello primero vamos a construir la función y luego ya la implementaremos en el archivo.

Esta función lo que tiene que hacer es recoger la votación del usuario, actualizar en la base de datos el campo de votaciones y mostrar un mensaje al usuario dándole las gracias por votar.

Os pongo la función construida y luego pasamos a explicarla.

```

function votar($id_articulo, $votacion){
    global $raiz;
    $respuesta = new xajaxResponse('ISO-8859-1');
    //echo "prueba";

    //insertamos el voto
    $ssql= "insert into articulo_voto (votacion, id_articulo) values ($votacion, $id_articulo)";

    //echo $ssql;
    mysql_query($ssql);

    //Saco el número de votos de este artículo
    $ssql_voto="select count(id_voto) as 'cuantos' from articulo_voto where id_articulo=$id_articulo";
    $rs voto=mysql_query($ssql voto);
    $fila voto=mysql_fetch_object($rs voto);

    $act="<b>Número de votos</b>:". $fila_voto->cuantos."<br>";

    //saco la votación media del artículo
    $ssql_media="select avg(votacion) as media from articulo_voto where id_articulo=".$id_articulo;
    $rs_media=mysql_query($ssql_media);
    $fila_media=mysql_fetch_object($rs_media);
    $act .="<b>Votación media</b>: <br>";
    if (!is_null($fila_media->media)){
        $absoluto=intval($fila_media->media);
        $decimal=$fila_media->media-$absoluto;
        if($absoluto>0){
            for ($x = 1; $x <= $absoluto; $x++){
                $act .="<img src='../images/estrella.gif' width=14 height=13>";
            }
        }
    }
}

```

```

        if($decimal>=0.5){
            $sact .='';
        }
    }else{
        $sact .="<i>Nadie ha votado por este artículo</i>";
    }
    $sact .="<br>";

    $respuesta->addAssign("votar_articulo","innerHTML",$res);
    $respuesta->addAssign("actualizar","innerHTML",$sact);
    return $respuesta;
}

```

Explicación del código

Lo primero que hacemos es insertar el voto en la base de datos con la primera sentencia sql.

Una vez introducido el voto pasamos a averiguar el número de votos realizados para luego mostrarlo en el recuadro de votación. Una vez creado el código que vamos a sustituir en el recuadro de votación averiguamos la votación media para mostrarla también en el recuadro añadiendo estrellitas (aquí puedes simplemente mostrar el numero resultante y sería más fácil de realizar). Si nadie ha votado por el artículo mostraríamos un mensaje diciendo que nadie ha votado todavía.

Para finalizar asignamos a la respuesta de la función las dos variables que hemos utilizado para almacenar lo que queremos mostrar en el recuadro de votación.

A continuación ponemos el código completo para que veáis como quedaría.

```

<?
require ($raiz . '../../../xajax/xajax.inc.php');
//instanciamos el objeto de la clase xajax
$xajax = new xajax();
$xajax->setCharEncoding('ISO-8859-1');
$xajax->decodeUTF8InputOn();
//conecto con la base de datos
$conn = mysql_connect("servidor","usuario","clave");
//selecciono la BBDD
mysql_select_db("bbdd",$conn);

function votar($id_articulo, $votacion){
    global $raiz;
    $respuesta = new xajaxResponse('ISO-8859-1');
    //echo "prueba";

    //insertamos el voto
    $ssql= "insert into articulo voto (votacion, id articulo) values ($votacion,
$id_articulo)";

    //echo $ssql;
    mysql_query($ssql);

    //Saco el número de votos de este articulo
    $ssql_voto="select count(id voto) as 'cuantos' from articulo voto where
id_articulo=$id_articulo";
    $rs_voto=mysql_query($ssql_voto);
    $fila_voto=mysql_fetch_object($rs_voto);

    $sact="<b>Número de votos</b>: ".$fila_voto->cuantos."<br>";

    //saco la votacion media del articulo
    $ssql_media="select avg(votacion) as media from articulo_voto where
id_articulo=".$id_articulo;
    $rs_media=mysql_query($ssql_media);
    $fila_media=mysql_fetch_object($rs_media);
    $sact .="<b>Votación media</b>: <br>";
    if (!is null($fila_media->media)){
        $absoluto=intval($fila_media->media);
    }
}

```

```

        $decimal=$fila_media->media-$absoluto;
        if($absoluto>0){
            for ($x = 1; $x <= $absoluto; $x++){
                $sact .='';
            }
            if($decimal>=0.5){
                $sact .='';
            }
        }
    }else{
        $sact .="<i>Nadie ha votado por este artículo</i>";
    }
    $sact .="<br>";

    $respuesta->addAssign("votar articulo","innerHTML",$res);
    $respuesta->addAssign("actualizar","innerHTML",$sact);
    return $respuesta;
}

//asociamos la función creada anteriormente al objeto xajax
$xajax->registerFunction("votar");
//El objeto xajax tiene que procesar cualquier petición
$xajax->processRequests();
$xajax->printJavascript("../../../../xajax/");
?>
<script>
function votar_articulo(id_articulo){

    var puntos = document.votar.puntos.value;
    xajax_votar(id_articulo,puntos)

}

</script>

<div style="width:192px;font-size:12pt; margin-top:3px;">
<div ><b>Participa:</b></div>
<table cellspacing="2" cellpadding="2" border="0" width="190">
<tr>
<td width="130" align=center class=fuente8>
    <div id="votar articulo">
        <?
        /*if ($respuesta!="")
            echo " <i>$respuesta</i> <br> ";*/
        //muestra media votos pagina($id breve);
        ?>

        Puntua el artículo:
        <br>
        <form action="sistema votacion.php" method="post" name=votar style="margin:0px;">
        <select name="puntos">
        <option value=1>1
        <option value=2>2
        <option value=3>3
        <option value=4>4
        <option value=5>5
        </select>
        <input type='button' name='votar' value='Votar' onClick='votar_articulo(1) '>
        </form>

    </div>
    </div>
    </td>
</tr>
<tr><td class=fuente8 align=center><div id=actualizar></div></td></tr>
</table>

</div>

```


Como podréis comprobar actualmente no es muy optimo el sistema ya que alguien podría votar varias veces seguidas.

Sistema de votación con Xajax mejorado

Esta vez vamos a comprobar si el que vota ya ha votado anteriormente, para ello utilizaremos las ips, por lo que tenemos que crear otra tabla en nuestra base de datos.

La tabla la vamos a llamar `articulo_ip` y va a tener el siguiente aspecto:

	Campo	Tipo
<input type="checkbox"/>	id_articulo	int(9)
<input type="checkbox"/>	ip	varchar(15)
<input type="checkbox"/>	fecha	int(14)

Una vez que tenemos la tabla vamos a modificar la función que tenemos creada llamada votar. Lo primero es crear unas variables de configuración al inicio de la función.

```
//CONFIGURACION
$tabla_control='articulo_ip';//tabla que controla las ip de las votaciones
$identificador='id_articulo';//nombre del campo en la tabla que controle las votaciones
$tabla_votos='articulo_voto';//tabla con el log de los votos
$id_cookie='rating_articulo';
$dias_votar_misma_ip = 2;
```

Después tenemos que crearnos una función que nos devuelva la ip real del visitante para ello utilizaremos esta función que ya tenemos construida y comentada.

```
function getRealIP(){
if( $ _SERVER['HTTP_X_FORWARDED_FOR'] != '' ){
    $client_ip =( !empty($ _SERVER['REMOTE_ADDR']) ) ?
        $ _SERVER['REMOTE_ADDR']:
        ( ( !empty($ ENV['REMOTE_ADDR']) ) ?
            $ ENV['REMOTE_ADDR']:
            "unknown" );

    // los proxys van añadiendo al final de esta cabecera
    // las direcciones ip que van "ocultando". Para localizar la ip real
    // del usuario se comienza a mirar por el principio hasta encontrar
    // una dirección ip que no sea del rango privado. En caso de no
    // encontrarse ninguna se toma como valor el REMOTE_ADDR

    $entries = split('[, ]', $ _SERVER['HTTP_X_FORWARDED_FOR']);

    reset($entries);
    while (list(, $entry) = each($entries)){
        $entry = trim($entry);
        if ( preg_match("/^([0-9]+.[0-9]+.[0-9]+.[0-9]+)/", $entry, $ip_list) ){
            // http://www.faqs.org/rfcs/rfc1918.html
            $private_ip = array(
                '/^0./',
                '/^127.0.0.1/',
                '/^192.168../',
                '/^172.((1[6-9])|(2[0-9])|(3[0-1]))../',
                '/^10../');

            $found_ip = preg_replace($private_ip, $client_ip, $ip_list[1]);

            if ($client_ip != $found_ip){
                $client_ip = $found_ip;
                break;
            }
        }
    }
}
```

```

    }
  }
} else {
  $client ip = ( !empty($ SERVER['REMOTE_ADDR']) ) ?
    $ _SERVER['REMOTE_ADDR']:
    ( ( !empty($ _ENV['REMOTE_ADDR']) ) ?
      $ _ENV['REMOTE_ADDR']:
      "unknown" );
}
return $client ip;
}

```

Una vez que tenemos la función la añadimos a nuestro archivo antes de la función de votar para así poder trabajar con ella y pasamos a modificar la función votar comprobando si hay cookies creadas o si tenemos la ip del usuario en nuestra tabla almacenada. Para ello creamos el siguiente código.

```

//controlamos que vote una sola vez
//controlamos si hay una cookie que impida la votacion
$nombre_cookie=$id_cookie.'_'. $id_articulo;
if ($ COOKIE[$nombre cookie]){
  //si hay cookie no permitimos la votacion
  $res="Ya has votado";
  //return $salida = false;
} else { //si no hay cookie
  //miramos si muestra ip
  $ip=getRealIP();

  if (!$ip){
    $res= "Incapaz de contabilizar";
    //return $salida = false; //si no la muestra la ip, no permitimos el voto
  } else {
    $segundos votar misma ip = 86400 * $dias votar misma ip;

    //borramos los registros de los dias de antigüedad que deseamos
    $ssql= "delete from articulo_ip where fecha+$segundos_votar_misma_ip <
unix_timestamp()";
    $rs = mysql query($ssql);

    //miramos si esta IP ya ha votado este recurso ultimamente (1 dia)
    $ssql= "select * from articulo_ip where (id_articulo=$id_articulo and ip =
'$ip' and unix_timestamp()-fecha<$segundos_votar_misma_ip)";
    //echo $ssql;
    $rs = mysql_query($ssql);

    //si no ha votado lo incorporamos en la tabla de control y creamos el voto
    if (mysql_num_rows($rs) == 0) {
      //metemos la ip en la tabla de control de voto
      $ssql= "insert into articulo_ip (id articulo, ip, fecha) values
($id articulo, '$ip', unix_timestamp())";
      mysql query($ssql);

      //insertamos el voto
      $ssql= "insert into articulo_voto (votacion, id articulo, ) values
($votacion, $id breve)";
      //echo $ssql;
      mysql_query($ssql);
      mysql_free_result($rs);

      //colocamos una cookie
      //setcookie($nombre cookie, time(), time() + $segundos votar misma ip);
      $res="Gracias por tu voto";
      //return true;
    } else {
      $res="Ya has votado";
      //return false;
    }
  }
}

```

Este código esta explicado para que vayáis viendo que hemos en cada momento pero básicamente lo que hace es, primero comprueba que no haya ninguna cookie, si no la hay pues

pasa a comprobar si hay algún registro en la tabla de ips, y si no lo hay inserta el voto, sino nos muestra un mensaje de que ya ha votado.

A continuación pongo el código completo de la función para que os quede del todo claro.

```
<?
require ($raiz . '../../xajax/xajax.inc.php');

//instanciamos el objeto de la clase xajax
$xajax = new xajax();
$xajax->setCharEncoding('ISO-8859-1');
$xajax->decodeUTF8InputOn();
//conecto con la base de datos
$conn = mysql_connect("servidor","usuario","contraseña");
//selecciono la BBDD
mysql_select_db("base de datos",$conn);
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////
//OBTIENE Y DEVUELVE LA IP REAL DE UN VISITANTE
////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////
function getRealIP()
{

if( $ SERVER['HTTP X FORWARDED FOR'] != '' )
{
$client_ip =
( !empty($ _SERVER['REMOTE_ADDR']) ) ?
$ _SERVER['REMOTE_ADDR']
:
( ( !empty($ ENV['REMOTE_ADDR']) ) ?
$ ENV['REMOTE_ADDR']
:
"unknown" );

// los proxys van añadiendo al final de esta cabecera
// las direcciones ip que van "ocultando". Para localizar la ip real
// del usuario se comienza a mirar por el principio hasta encontrar
// una dirección ip que no sea del rango privado. En caso de no
// encontrarse ninguna se toma como valor el REMOTE_ADDR

$entries = split('[, ]', $ SERVER['HTTP X FORWARDED FOR']);

reset($entries);
while (list(, $entry) = each($entries))
{
$entry = trim($entry);
if ( preg_match("/^([0-9]+.[0-9]+.[0-9]+.[0-9]+)/", $entry, $ip list) )
{
// http://www.faqs.org/rfcs/rfc1918.html
$private_ip = array(
'^0./',
'^127.0.0.1/',
'^192.168../',
'^172.( (1[6-9]) | (2[0-9]) | (3[0-1]))../',
'^10../' );

$found_ip = preg_replace($private_ip, $client_ip, $ip_list[1]);

if ($client_ip != $found_ip)
{
$client_ip = $found_ip;
break;
}
}
}
else
{
$client_ip =
( !empty($ SERVER['REMOTE_ADDR']) ) ?
$ _SERVER['REMOTE_ADDR']
:
( ( !empty($ ENV['REMOTE_ADDR']) ) ?
```

```

$_ENV['REMOTE_ADDR']
:
"unknown" );
}
return $client_ip;
}
function votar($id_articulo, $votacion){
    global $raiz;
    $respuesta = new xajaxResponse('ISO-8859-1');
    //echo "prueba";
    //CONFIGURACION
    $tabla_control='articulo_ip';//tabla que controla las ip de las votaciones
    $identificador='id articulo';//nombre del campo en la tabla que controle las
votaciones
    $tabla_votos='articulo voto';//tabla con el log de los votos
    $id_cookie='rating articulo';
    $dias_votar_misma_ip = 2;

    //controlamos que vote una sola vez
    //controlamos si hay una cookie que impida la votacion
    $nombre_cookie=$id_cookie.' '.$id_articulo;
    if ($ COOKIE[$nombre_cookie]){
        //si hay cookie no permitimos la votacion
        $res="Ya has votado";
        //return $salida = false;
    }else{//si no hay cookie
        //miramos si muestra ip
        $ip=getRealIP();

        if(!$ip){
            $res= "Incapaz de contabilizar";
            //return $salida = false;//si no la muestra la ip, no permitimos el voto
        }else{
            $segundos_votar_misma_ip = 86400 * $dias_votar_misma_ip;

            //borramos los registros de los dias de antigüedad que deseamos
            $ssql= "delete from articulo_ip where fecha+$segundos_votar_misma_ip <
unix timestamp()";
            $rs = mysql_query($ssql);

            //miramos si esta IP ya ha votado este recurso ultimamente (1 dia)
            $ssql = "select * from articulo ip where (id articulo=$id articulo and ip =
'$ip' and unix timestamp()-fecha<$segundos votar misma ip)";
            //echo $ssql;
            $rs = mysql_query($ssql);

            //si no ha votado lo incorporamos en la tabla de control y creamos el voto
            if (mysql_num_rows($rs) == 0) {
                //metemos la ip en la tabla de control de voto
                $ssql = "insert into articulo_ip (id_articulo, ip, fecha) values
($id_articulo, '$ip', unix_timestamp())";
                mysql_query($ssql);

                //insertamos el voto
                $ssql= "insert into articulo voto (votacion, id articulo, ) values
($votacion, $id breve)";
                //echo $ssql;
                mysql_query($ssql);
                mysql_free_result($rs);

                //colocamos una cookie
                //setcookie($nombre_cookie, time(), time() + $segundos_votar_misma_ip);
                $res="Gracias por tu voto";
                //return true;
            }else{
                $res="Ya has votado";
                //return false;
            }
        }
    }

    //Saco el número de votos de este articulo
    $ssql voto="select count(id voto) as 'cuantos' from articulo voto where
id articulo=$id articulo";
    $rs_voto=mysql_query($ssql_voto);
    $fila_voto=mysql_fetch_object($rs_voto);

```

```

    $act="<b>Número de votos</b>:".$fila_voto->cuantos."<br>";

    //saco la votacion media del articulo
    $ssql media="select avg(votacion) as media from articulo voto where
id_articulo=".$id_articulo;
    $rs_media=mysql_query($ssql_media);
    $fila_media=mysql_fetch_object($rs_media);
    $act .="<b>Votación media</b>: <br>";
    if (!is_null($fila_media->media)){
        $absoluto=intval($fila_media->media);
        $decimal=$fila_media->media-$absoluto;
        if($absoluto>0){
            for ($x = 1; $x <= $absoluto; $x++){
                $act .="<img src='../.../images/estrella.gif" width=14 height=13>';
            }
            if($decimal>=0.5){
                $act .="<img src='../.../images/estrellamitad.gif" width=14 height=13>';
            }
        }
    }else{
        $act .="<i>Nadie ha votado por este artículo</i>";
    }
    $act .="<br>";

    $respuesta->addAssign("votar articulo","innerHTML",$res);
    $respuesta->addAssign("actualizar","innerHTML",$act);
    return $respuesta;
}

//asociamos la función creada anteriormente al objeto xajax
$xajax->registerFunction("votar");
//El objeto xajax tiene que procesar cualquier petición
$xajax->processRequests();
$xajax->printJavascript("../.../xajax/");
?>
<script>
function votar articulo(id articulo){

    var puntos = document.votar.puntos.value;
    xajax_votar(id_articulo,puntos)

}

</script>

<div style="width:192px;font-size:12pt; margin-top:3px;">
<div ><b>Participa:</b></div>
<table cellpadding="2" cellspacing="2" border="0" width="190">
<tr>
<td width="130" align=center class=fuente8>
    <div id="votar articulo">
        <?
        /*if ($respuesta!="")
            echo " <i>$respuesta</i> <br> ";*/
        //muestra_media_votos_pagina($id_breve);
        ?>

Puntua el artículo:
<br>
<form action="sistema_votacion.php" method="post" name=votar style="margin:0px;">
<select name="puntos">
<option value=1>1
<option value=2>2
<option value=3>3
<option value=4>4
<option value=5>5
</select>
<input type='button' name='votar' value='Votar' onClick='votar articulo(1)''>
</form>

</div>
</div>
</td>

```

```

</tr>
<tr><td class=fuente8 align=center><div id=actualizar></div></td></tr>
</table>

</div>

```

Incluir un archivo de texto o HTML a través de una llamada a xajax

La función es bastante sencilla tan solo tenemos que pasarle por parámetro el nombre del archivo y el id de la capa donde queremos colocar el contenido del archivo. De esta forma la cabecera de la función sería algo así.

```
function anadir_codigo($archivo,$id){
```

Ahora pasamos a escribir el código de la función:

Para ello lo primero que tenemos que hacer es meter una variable global con la raíz para que busque bien los archivos. Y después tan solo tenemos que abrir el archivo y pasarlo a una variable la cual es el valor devuelto por la función.

```

global $raiz;
$respuesta = new xajaxResponse('ISO-8859-1');
$path = $raiz . $archivo;
$fp = fopen ($path,'r');
$codigo="";
while ($linea = fgets($fp,1024))
{
    if ($linea) $codigo .= $linea;
}
fclose($fp);
//$codigo=htmlspecialchars($codigo);
//$codigo=htmlentities($codigo);

//$codigo=str_replace(">",">",$codigo);
$res=$codigo;
//echo $res;
$respuesta->addAssign($id,"innerHTML",$res);
//echo $res;
return $respuesta;

```

De esta forma tenemos la función xajax creada. Ya solo nos queda incluirla en un contexto para que veáis como funciona.

Para ello vamos a crear un archivo de ejemplo. Dejo el código completo y a continuación lo explicamos.

```

<?
$raiz = "../..../"; //empieza en . termina en /
//include ("../librerias/principales.php");
require ($raiz. 'xajax/xajax.inc.php');
//instanciamos el objeto de la clase xajax
$xajax = new xajax();
$xajax->setCharEncoding('ISO-8859-1');
$xajax->decodeUTF8InputOn();

function anadir_codigo($archivo,$id){
global $raiz;
$respuesta = new xajaxResponse('ISO-8859-1');
$path = $raiz . $archivo;
$fp = fopen ($path,'r');
$codigo="";
while ($linea = fgets($fp,1024))
{
    if ($linea) $codigo .= $linea;
}

```

```

    }
    fclose($fp);
    $res=$codigo;
    $respuesta->addAssign($id,"innerHTML",$res);

    return $respuesta;
}
//registramos la función creada anteriormente al objeto xajax
$xajax->registerFunction("anadir_codigo");

//El objeto xajax tiene que procesar cualquier petición
$xajax->processRequests();

$xajax->printJavascript($raiz . "xajax/");
?>
<html>
<head>
<script>
function llamar_codigo(archivo,id capa){
    xajax anadir_codigo(archivo,id capa)

}

</script>
</head>
<body>
<div id="mensaje">Si pulsamos <a
href="javascript:llamar_codigo('articulos/ejemplos/ajax/codigo_formulario.html',
'mensaje')">aquí</a> nos va cambiar esto por lo que tengamos en el código del archivo
que le pasamos.</div>
</body>

```

Lo único que hemos construido aquí es crear un archivo donde hemos colocado la función xajax y la hemos llamado desde un simple div para sustituir el texto por un pequeño formulario.

El código que contiene el archivo código_formulario.html es el siguiente:

```

<form action="#" method="post">
    <input type="text" name="nombre" value="nombre">
    <input type="submit" name="enviar" value="enviar">
</form>

```

Como podéis comprobar es muy sencillito pero resulta realmente útil a la hora de crear nuestras páginas web.

Mensaje de carga con ajax usando xajax

Vamos a ver una de las utilidades de xajax para mostrar un mensaje de carga, para advertir al usuario mientras se procesa una solicitud ajax. Como habremos visto en casi todas las aplicaciones que utilizan Ajax, cuando se está procesando una solicitud, suele aparecer un mensaje de carga, para que el usuario sepa que se está llevando a cabo su solicitud, mientras que los resultados no se muestran en la página. Este es un punto muy interesante, porque a menudo los procesamientos de Ajax tardan en presentarse al usuario y mientras tanto, si no se le informa que se está cargando la página, puede parecer que no se está efectuando ninguna acción.

Este artículo se engloba dentro del [manual de Ajax en PHP utilizando Xajax](#). Es importante que se sepa que hasta el momento en este manual estamos tratando siempre la versión 0.2.5 de Xajax, que es la que está publicada como estable y, por tanto, es la que se aconseja en el momento de escribir este artículo para entornos en producción. Para versiones posteriores de Xajax (en concreto la 0.5) estas indicaciones cambian, pero podéis ver cómo hacerlo en [Update de script a Xajax 0.5](#).

Primero veamos una función de PHP que se invocaría por Ajax que tardaría en procesarse un cierto tiempo:

```
function funcion_lenta()
{
    //retenemos el procesamiento por 3 segundos
    sleep(3);

    $objResponse = new xajaxResponse();
    $objResponse->addAssign("capa_actualiza","innerHTML","Finalizado");

    return $objResponse;
}
```

Esta función, que sería la más simple que se podría hacer, tardará al menos 3 segundos en procesarse. Para asegurarnos de ello hemos hecho que el procesamiento de PHP se detenga por 3 segundos con la función sleep(3).

Para llamar a esta función por medio de xajax haremos algo como esto:

```
<a href="javascript:void(xajax_funcion_lenta())">Llamar por ajax a una función lenta usando xajax</a>!
```

Nota: hay una serie de líneas de código que serían necesarias para que esto funcionase con xajax, como el processRequests() o el printJavascript(), que no estoy comentando porque hemos visto en capítulos anteriores del manual. De todos modos muestro el código completo de este ejemplo previo, que es sólo una prueba para enseñar el efecto producido por una espera en el procesamiento ajax.

```
<?
require ('xajax/xajax.inc.php');

function funcion_lenta()
{
    sleep(3);

    $objResponse = new xajaxResponse();
    $objResponse->addAssign("capa_actualiza","innerHTML","Finalizado");

    return $objResponse;
}

$xajax = new xajax();
$xajax->registerFunction('funcion_lenta');
$xajax->processRequests();

?>
<html>
<head>
    <title>Ejemplo de mostrar un aviso de carga de la página</title>
    <?
    $xajax->printJavascript("xajax/");
    ?>
</head>
<body>
<div id="capa_actualiza">
<a href="javascript:void(xajax_funcion_lenta())">Llamar por ajax a una función lenta
```



```
usando xajax</a>!  
</div>  
</body>  
</html>
```

Código para mostrar un mensaje de carga con Xajax

Xajax tiene unas utilidades específicas en la versión 0.2.5 para mostrar los mensajes de "Cargando" que vamos a utilizar, que nos van a simplificar bastante la vida.

Para crear el mensaje de carga primero debemos colocar una capa donde aparecerá el texto "Cargando..." o la típica imagen de carga, esa que es como una rueda que va moviéndose.

```
<div id="MensajeCarga" style="display: none;">  
Cargando!...  
</div>
```

Hay que señalar que la capa tiene el estilo `display: none;` para que no se muestre en la página, en un principio y hasta que no lo indiquemos con xajax.

Ahora, en el framework Xajax simplemente tenemos que especificar con Javascript dos propiedades del objeto xajax, que se ha instanciado implícitamente al hacer el `$xajax->printJavascript()`. Son las siguientes:

```
xajax.loadingFunction = [function para mostrar el mensaje de carga];  
xajax.doneLoadingFunction = [function para ocultar el mensaje de carga];
```

A ambas propiedades debemos asignarles como valores sendas funciones javascript que serán las encargadas de mostrar y ocultar el mensaje de carga. Esto lo podremos hacer con un código como este, que tendremos que colocar después del `printJavascript()`.

Para actualizar este código para Xajax 0.5. En la versión Xajax 0.5, para indicar lo que se debe hacer al iniciar la solicitud por Ajax y al completarla, se deben utilizar otras propiedades de otros objetos:

```
xajax.callback.global.onResponseDelay = muestra_cargando;  
xajax.callback.global.onComplete = oculta_cargando;  
  
<script type="text/javascript">  
<!--  
    function muestra_cargando(){  
        xajax.$('MensajeCarga').style.display='block';  
    }  
    function oculta_cargando(){  
        xajax.$('MensajeCarga').style.display='none';  
    }  
  
xajax.loadingFunction = muestra_cargando;  
xajax.doneLoadingFunction = oculta_cargando;  
// --></script>
```

Como se puede ver, hemos creado dos funciones `muestra_cargando()` y `oculta_cargando()`, a las que hemos colocado dos códigos para cambiar la propiedad `display` de los estilos CSS de la capa "MensajeCarga". En `muestra_cargando()` hacemos `display='block'` para que se muestre la capa y en `oculta_cargando()` hacemos `display='none'` para ocultarla.

Luego hemos asignado esas funciones a `xajax.loadingFunction` y `xajax.doneLoadingFunction`, tal como habíamos especificado.

Eso es todo! Ahora se invocará `xajax.loadingFunction` automáticamente cuando se realice una llamada a Ajax por medio de Xajax. También automáticamente se invocará a `xajax.doneLoadingFunction` cuando se termine de procesar la solicitud Ajax.

El código PHP del ejemplo completo lo podemos ver a continuación:

```
<?
require ('xajax/xajax.inc.php');

function funcion lenta(){
    sleep(3);

    $objResponse = new xajaxResponse();
    $objResponse->addAssign("capa_actualiza","innerHTML","Finalizado");

    return $objResponse;
}

$xajax = new xajax();
$xajax->registerFunction('funcion_lenta');
$xajax->processRequests();

?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>
<head>
    <title>Ejemplo de mostrar un aviso de carga de la página</title>
    <?
    $xajax->printJavascript("xajax/");
    ?>
    <script type="text/javascript">
<!--
    function muestra_cargando(){
        xajax.$('MensajeCarga').style.display='block';
    }
    function oculta_cargando(){
        xajax.$('MensajeCarga').style.display='none';
    }

    xajax.loadingFunction = muestra_cargando;
    xajax.doneLoadingFunction = oculta_cargando;
// --></script>
</head>

<body>

<div id="capa actualiza">
<a href="javascript:void(xajax funcion lenta())">Llamar por ajax a una función lenta
usando xajax</a>!
</div>

<div id="MensajeCarga" style="display: none;">
Cargando!...
</div>

</body>
</html>
```

Sistema de ordenación de elementos

Vamos a crear un ordenador de elementos con Ajax y PHP. Esto es, un sistema para cambiar el orden de los registros de una tabla de una base de datos con funciones Xajax, una librería para hacer Ajax cómodamente en PHP. Como utilizamos Ajax, el ordenador de elementos funcionará dinámicamente, sin necesidad de que se recargue la página.

A continuación veamos los pasos para la creación de este sistema de ordenación de elementos.

Primero mostramos los datos de la tabla ordenados por el campo orden de la base de datos. En este caso, hemos creado unos botones para poder cambiar el orden de los elementos que queremos ordenar.

```
$ssql="select * from color order by orden";
$rs=mysql_query($ssql);
$cont=mysql_num_rows($rs);
$num=1;
echo 'ordenar colores<br>
<br>
';
echo '<div id="colores">';
while($fila=mysql_fetch_object($rs)){
if($fila->orden==0){
    echo'

        <input type="Button" name="orden" value="Bajar" onclick="xajax_aumentar('.$fila-
>id color.','.$fila->orden.');">
        '.$fila->nombre color;
    }elseif($num==$cont){
        echo
        <input type="Button" name="orden" value="Subir" onclick="xajax_disminuir('.$fila-
>id_color.','.$fila->orden.');">

        '.$fila->nombre_color;
    }else{
        echo '
        <input type="Button" name="orden" value="Subir" onclick="xajax_disminuir('.$fila-
>id_color.','.$fila->orden.');">
        <input type="Button" name="orden" value="Bajar" onclick="xajax_aumentar('.$fila-
>id color.','.$fila->orden.');">
        '.$fila->nombre_color;
    }

    echo '<br>';

    $num++;
}
echo '</div>';
```

Las funciones reciben el id del elemento que se quiere cambiar de orden y la posición que tiene actualmente, actualiza los registros de la tabla y vuelve a mostrar los elementos en el orden actualizado.

```
function aumentar($id_color,$orden){
    $orden_siguiente=$orden+1;
    //actualizo el registro actual
    $ssql="update color set orden=".$orden." where orden=".$orden_siguiente;
    $rs=mysql_query($ssql);
    //actualizo el registro siguiente
    $ssql="update color set orden=orden+1 where id color=".$id_color;
    $rs=mysql_query($ssql);
    //muestro todos los registros actualizados
    $ssql="select * from color order by orden";
    $rs=mysql_query($ssql);
    $cont=mysql_num_rows($rs);
    $num=1;
    while($fila=mysql_fetch_object($rs)){
        if($fila->orden==0){
            $contenido.=' '
            $contenido.='<input type="Button" name="orden" value="Bajar"
onclick="xajax_aumentar('.$fila->id_color.','.$fila->orden.');">';
            $contenido.=' '.$fila->nombre_color;
        }elseif($num==$cont){
            $contenido.='<input type="Button" name="orden" value="Subir"
onclick="xajax_disminuir('.$fila->id_color.','.$fila->orden.');">';
            $contenido.=' '
            $contenido.=' '.$fila->nombre_color;
        }else{

```

```

        $contenido.='<input          type="Button"          name="orden"          value="Subir"
onclick="xajax_disminuir('.$fila->id_color.','.$fila->orden.');">';
        $contenido.='<input          type="Button"          name="orden"          value="Bajar"
onclick="xajax_aumentar('.$fila->id_color.','.$fila->orden.');">';
        $contenido.=$fila->nombre_color;
    }
    $num++;
    $contenido.='<br>';
}

$respuesta = new xajaxResponse('ISO-8859-1');
$respuesta->addAssign("colores","innerHTML",$contenido);
return $respuesta;
}

```

Lo primero que hace la función es actualizar el registro que queremos ordenar y cambia el orden del registro anterior o posterior al registro actualizado, dependiendo de si lo que queremos es aumentar o disminuir una posición. Por último muestra los registros de la tabla en el orden actualizado.

A continuación mostramos el código completo para ver cómo quedaría.

```

<?
require ($raiz . '../../xajax/xajax.inc.php');
//instanciamos el objeto de la clase xajax
$xajax = new xajax();
$xajax->setCharEncoding('ISO-8859-1');
$xajax->decodeUTF8InputOn();
//conecto con la base de datos
$conn = mysql_connect("servidor","usuario","clave");
//selecciono la BBDD
mysql_select_db("bbdd",$conn);

function aumentar($id_color,$orden){
    $orden_siguiente=$orden+1;
    //actualizo el registro actual
    $ssql="update color set orden=".$orden." where orden=".$orden_siguiente;
    $rs=mysql_query($ssql);
    //actualizo el registro siguiente
    $ssql="update color set orden=orden+1 where id color=".$id_color;
    $rs=mysql_query($ssql);
    //muestro todos los registros actualizados
    $ssql="select * from color order by orden";
    $rs=mysql_query($ssql);
    $cont=mysql_num_rows($rs);
    $num=1;
    while($fila=mysql_fetch_object($rs)){
        if($fila->orden==0){
            $contenido.=' ';
            $contenido.='<input          type="Button"          name="orden"          value="Bajar"
onclick="xajax_aumentar('.$fila->id_color.','.$fila->orden.');">';
            $contenido.=$fila->nombre_color;
        }elseif($num==$cont){
            $contenido.='<input          type="Button"          name="orden"          value="Subir"
onclick="xajax_disminuir('.$fila->id_color.','.$fila->orden.');">';
            $contenido.=' ';
            $contenido.=$fila->nombre_color;
        }else{
            $contenido.='<input          type="Button"          name="orden"          value="Subir"
onclick="xajax_disminuir('.$fila->id_color.','.$fila->orden.');">';
            $contenido.='<input          type="Button"          name="orden"          value="Bajar"
onclick="xajax_aumentar('.$fila->id_color.','.$fila->orden.');">';
            $contenido.=$fila->nombre_color;
        }
        $num++;
        $contenido.='<br>';
    }

    $respuesta = new xajaxResponse('ISO-8859-1');
    $respuesta->addAssign("colores","innerHTML",$contenido);
    return $respuesta;
}

//asociamos la función creada anteriormente al objeto xajax

```

```

$xajax->registerFunction("aumentar");

function disminuir($id_color,$orden){
    $orden anterior=$orden-1;
    //actualizo el registro actual
    $ssql="update color set orden=".$orden." where orden=".$orden_anterior;
    $rs=mysql_query($ssql);
    //actualizo el registro anterior
    $ssql="update color set orden=orden-1 where id_color=".$id_color;
    $rs=mysql_query($ssql);
    //muestro los datos actualizados
    $ssql="select * from color order by orden";
    $rs=mysql_query($ssql);
    $cont=mysql_num_rows($rs);
    $num=1;
    while($fila=mysql_fetch_object($rs)){
        if($fila->orden==0){
            $contenido.=' ';
            $contenido.='<input type="Button" name="orden" value="Bajar"
onclick="xajax_aumentar('.$fila->id_color.','.$fila->orden.');">';
            $contenido.=$fila->nombre_color;
        }elseif($num==$cont){
            $contenido.='<input type="Button" name="orden" value="Subir"
onclick="xajax_disminuir('.$fila->id_color.','.$fila->orden.');">';
            $contenido.=' ';
            $contenido.=$fila->nombre_color;
        }else{
            $contenido.='<input type="Button" name="orden" value="Subir"
onclick="xajax_disminuir('.$fila->id_color.','.$fila->orden.');">';
            $contenido.='<input type="Button" name="orden" value="Bajar"
onclick="xajax_aumentar('.$fila->id_color.','.$fila->orden.');">';
            $contenido.=$fila->nombre_color;
        }
        $num++;
        $contenido.='<br>';
    }

    $respuesta = new xajaxResponse('ISO-8859-1');
    $respuesta->addAssign("colores","innerHTML",$contenido);
    return $respuesta;
}

//asociamos la función creada anteriormente al objeto xajax
$xajax->registerFunction("disminuir");

//El objeto xajax tiene que procesar cualquier petición
$xajax->processRequests();

//En el <head> indicamos al objeto xajax se encargue de generar el javascript necesario
$xajax->printJavascript("../../xajax/");

$ssql="select * from color order by orden";
$rs=mysql_query($ssql);
$cont=mysql_num_rows($rs);
$num=1;
echo '<b>Ordenar colores</b><br>
<br>
';
echo '<div id="colores">';
while($fila=mysql_fetch_object($rs)){
    if($fila->orden==0){
        echo '

        <input type="Button" name="orden" value="Bajar" onclick="xajax_aumentar('.$fila->id_color.','.$fila->orden.');">
        '.$fila->nombre_color;
    }elseif($num==$cont){
        echo
        '<input type="Button" name="orden" value="Subir" onclick="xajax_disminuir('.$fila->id_color.','.$fila->orden.');">

        '.$fila->nombre_color;
    }else{
        echo '
        <input type="Button" name="orden" value="Subir" onclick="xajax_disminuir('.$fila->id_color.','.$fila->orden.');">

```

```

        <input type="Button" name="orden" value="Bajar" onclick="xajax_aumentar('.$fila-
>id_color.','.$fila->orden.');">
        '.$fila->nombre color;
    }

    echo '<br>';

    $num++;
}
echo '</div>';
?>

```

Actualizar Xajax 0.2 a Xajax 0.5

Afortunadamente, en la propia web de Xajax han publicado una receta con una serie de referencias adicionales sobre métodos que han cambiado de nombre o de manera de acceder a ellos. Podemos ver esas referencias, aunque en inglés en esta página:

<http://xajaxproject.org/docs/upgrading-from-xajax-0.2-to-xajax-0.5.php>

No obstante, resumimos aquí los principales puntos.

Instalar Xajax 0.5

Como primera recomendación, en la página web de Xajax, nos informan que debemos hacer una copia de seguridad de la distribución de Xajax que tuviésemos trabajando anteriormente en nuestro servidor. Luego, debemos borrar los archivos y el directorio. En su lugar debemos poner luego los archivos de la nueva versión Xajax 0.5. Insisten en que se debe borrar la instalación antigua, y no instalar xajax 0.5 en ningún caso sobre el directorio donde tuviéramos las librerías de la versión anterior.

Actualizar los includes

Debemos revisar los `includes` para incluir Xajax en nuestros scripts PHP, puesto que la estructura de directorios del framework ha cambiado en la versión 0.5, para separar los códigos PHP de los códigos Javascript. Los archivos PHP están ahora en `xajax_core` y los Javascript en `xajax_js`.

3. Actualiza la sintaxis

La versión Xajax 0.5 RC1 es ya la versión definitiva que se va a presentar, de modo que, aunque se encuentre en Release Candidate aun, no va a revestir cambios significativos.

La mayoría de cambios de sintaxis que tendremos que hacer de cara a actualizar a la versión 0.5 son relativos a funciones, que se han quedado obsoletas y ahora llevan otros nombres o dependen de otras clases. Esas referencias debemos actualizarlas, puesto que en el futuro van a desaparecer las funciones marcadas como obsoletas.

Los métodos más destacables que tienen nuevos nombres:

Métodos de la clase xajax

El antiguo método `$xajax->processRequests()`; ahora se llama `$xajax->processRequest()`;

Todas las configuraciones booleanas como `$xajax->debugOn()`; y `$xajax->outputEntitiesOn()`; se han transformado en `$xajax->setFlag('debug',true)`; y `$xajax->setFlag('outputEntities',true)`;

```
$xajax->setFlags(); ahora también puede recibir arrays como $xajax->setFlags(
array('debug'=>true, 'outputEntities'=>true));
```

El método `$xajax->registerFunction` se ha marcado como obsoleto por la existencia de la nueva función `$xajax->register()`. Esta nueva función `register` recibe dos parámetros. El primero es el tipo de aquello que queremos registrar y el segundo es su nombre. Por ejemplo, para el caso de registrar una función se haría:

```
$xajax->register(XAJAX_FUNCTION, 'nombre_de_la_funcion');
```

Métodos de la clase `xajaxResponse`

La mayoría de los métodos de `xajaxResponse` han cambiado. Han suprimido una parte que se consideraba confusa en los nombres de métodos. Resulta que muchos comenzaban por `add` y se ha quitado. Por ejemplo `xajaxResponse->addAssign()` ahora es `xajaxResponse->assign()`.

Ejemplo de actualización de script Xajax

Ahora voy a hacer una actualización de un script que había creado anteriormente en el manual con Xajax 0.2 a Xajax 0.5. Es un script para hacer `selects` combinados que vimos en el artículo [Selects combinados con Ajax y PHP](#).

Simplemente voy a colocar aquí el código nuevo, marcando en negrita, para que queden bien claros los cambios que he realizado.

```
<?
//incluimos la clase ajax
require ('xajax/xajax_core/xajax.inc.php');

//instanciamos el objeto de la clase xajax
$xajax = new xajax();
$xajax->setCharEncoding('ISO-8859-1');
$xajax->configure('decodeUTF8Input', true);

function select_combinado($id_provincia){
    $madrid = array("Madrid", "Las Rozas", "Móstoles", "San Sebastián de los Reyes");
    $valencia = array("Valencia", "La Eliana", "Paterna", "Cullera");
    $barcelona = array("Barcelona", "Badalona");
    $leon = array ("León", "Astorga", "Villamejil");
    $poblaciones = array($madrid, $valencia, $barcelona, $leon);

    $nuevo_select = "<select name='poblaciones'>";

    for ($i=0; $i<count($poblaciones[$id_provincia]); $i++){
        //for ($i=0; $i<2; $i++){
            $nuevo_select .= '<option value="' . $i . '">' . $poblaciones[$id_provincia][$i] .
        '</option>';
        }
        $nuevo_select .= "</select>";
        return $nuevo_select;
    }

function generar_select($cod_provincia){
    $respuesta = new xajaxResponse();
    $respuesta->setCharacterEncoding('ISO-8859-1');

    if ($cod_provincia==999){
        $nuevo_select = '<select name="poblaciones">
            <option value=0>Elegir provincia</option>
        </select>';
    }else{
        $nuevo_select = select_combinado($cod_provincia);
    }
    $respuesta->Assign("seleccombinado", "innerHTML", $nuevo_select);

    return $respuesta;
}

$xajax->register(XAJAX_FUNCTION, 'generar_select');
```

```
//El objeto xajax tiene que procesar cualquier petición
$xajax->processRequest();
?>

<html>
<head>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=ISO-8859-1">
<title>Selects combinados en PHP y Ajax - Desarrolloweb.com</title>
<?
//Esta línea no cambia porque el printJavascript sigue recibiendo
//la ruta a la carpeta raíz donde están las librerías xajax.
$xajax->printJavascript("xajax/");
?>
</head>

<body>

<form name="formulario">
Provincia:
<br>
<select name="provincia"
onchange="xajax_generar_select(document.formulario.provincia.options[document.formulario
.provincia.selectedIndex].value)">
<option value="999">Selecciona la provincia</option>
<option value=0>Madrid</option>
<option value=1>Valencia</option>
<option value=2>Barcelona</option>
<option value=3>León</option>
</select>
<br>
<br>
Selecciona Población: <div id="seleccombinado">
<select name="poblaciones">
<option value=0>Elegir provincia</option>
</select>
</div>
</form>
Por Desarrolloweb.com!
</body>
</html>
```

Para una referencia completa de funciones que han cambiado de Xajax 0.2 a Xajax 0.5 podemos visitar el artículo que había comentado antes de la [documentación de Xajax](#), que contiene abajo del todo un listado de referencia rápida de funciones nuevas y antiguas.

Aviso de cargando para Ajax con Xajax 0.5

A continuación vamos a ver el código de [aviso de cargando para Ajax](#) para Xajax 0.5, poniendo en negrita el código modificado:

```
<?
require ('xajax/xajax_core/xajax.inc.php');

function funcion_lenta(){
    sleep(3);

    $objResponse = new xajaxResponse();
    $objResponse->Assign("capa_actualiza","innerHTML","Finalizado");

    return $objResponse;
}

$xajax = new xajax();
$xajax->register(XAJAX_FUNCTION, 'funcion_lenta');
$xajax->processRequest();

?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
```



```

<html>
  <head>
    <title>Ejemplo de mostrar un aviso de carga de la página</title>
    <?
      $xajax->printJavascript("xajax/");
    ?>
    <script type="text/javascript">
      <!--
        function muestra_cargando() {
          xajax.dom.create("capa_actualiza", "div", "cargando");
          xajax.$('cargando').innerHTML = '';
        }
        function oculta_cargando() {
          alert("cargado");
        }

        xajax.callback.global.onResponseDelay = muestra_cargando;
        xajax.callback.global.onComplete = oculta_cargando;
      // -->
    </script>
  </head>

  <body>

    <div id="capa_actualiza">
      <a href="javascript:void(xajax_funcion_lenta())">
        Llamar con ajax a una función que tarda en cargar
      </a>!
    </div>

  </body>
</html>

```

Hemos visto que los cambios no son muchos y es de agradecer. Pero vamos a verlos uno a uno, aunque recordamos que están explicados casi todos estos cambios en el artículo anterior.

Veamos pues, línea a línea, los cambios resaltados de este script.

```
require ('xajax/xajax_core/xajax.inc.php');
```

Esto ha cambiado porque ahora las librerías de Xajax están en carpetas distintas.

```
$objResponse->Assign("capa_actualiza", "innerHTML", "Finalizado");
```

El método `addAssign` del objeto de la clase `xajaxResponse` ahora se llama simplemente `Assign`.

```
$xajax->register(XAJAX_FUNCTION, 'funcion_lenta');
```

La función `registerFunction` ahora se llama `register` y debemos especificar qué es lo que estamos registrando.

```
$xajax->processRequest();
```

Esta función también ha cambiado el nombre. Antes era `$xajax->processRequests()` y ahora `$xajax->processRequest()` (*hemos quitado la "s" final*).

Ahora viene lo más interesante, que son las funciones que servirán para mostrar el mensaje de carga. Estas dos funciones que hemos creado sirven para realizar las acciones necesarias para mostrar y ocultar el mensaje de carga. Por darle algo más de interés a este artículo no nos hemos limitado a traducirlas a Xajax 0.5, sino que además hemos hecho un par de cambios en su comportamiento.

El cambio principal, que decíamos fue pedido por un lector, consiste en que el mensaje de carga aparece en la propia capa donde vamos a colocar el contenido traído por Ajax. Antes teníamos una capa con el mensaje de carga que simplemente mostrábamos al iniciar la carga y luego la ocultábamos al terminarse.

En este código hemos utilizado unas funciones de Xajax, de la librería de funciones Javascript llamadas `xajax_core.js`. Estas funciones sirven para realizar cosas con Javascript y en concreto hemos utilizado varias para alterar los contenidos del DOM de la página, es decir, los objetos del navegador que sirven para alterar dinámicamente los contenidos de la página.

Veamos el código del que estoy hablando:

```
function muestra_cargando(){
    xajax.dom.create("capa_actualiza","div", "cargando");
    xajax.$('cargando').innerHTML='';
}
function oculta_cargando(){
    alert("cargado");
}

xajax.callback.global.onResponseDelay = muestra_cargando;
xajax.callback.global.onComplete = oculta_cargando;
```

En la función `muestra_cargando` lo que hago es crear una nueva etiqueta `DIV` e insertarla en la capa que se va a actualizar con los contenidos de la solicitud Ajax. Esto se hace con la función `xajax.dom.create()`, que recibe la capa donde se va a crear el nuevo elemento HTML, el nombre del elemento o etiqueta que se va a crear y el identificador que se va a asignar a dicho elemento creado.

Luego con `xajax.$('cargando').innerHTML` lo que se consigue es asignar un código HTML a la capa recién creada. `xajax.$('cargando')` es una referencia a la capa que se acaba de crear y con su propiedad `innerHTML` accedemos al código HTML que tiene dentro. Simplemente asignamos un código para mostrar una imagen.

En la siguiente función `oculta_cargando()`, simplemente muestro un mensaje de alerta para que se sepa que se ha terminado de cargar. Esta función no hacía mucha falta en este ejemplo, puesto que la solicitud Xajax realizada por la función PHP `funcion_lenta()` iba a sustituir el contenido de la "capa actualiza" y por tanto desaparecerán con ellos la imagen de carga generada con la función `muestra_cargando()`. No obstante, la he colocado para que sirva de ejemplo y por si el lector la quiere utilizar para cualquier cosa que necesite.

Por último las líneas:

```
xajax.callback.global.onResponseDelay = muestra_cargando;
xajax.callback.global.onComplete = oculta_cargando;
```

Simplemente hacen que se asocien este par de funciones a las solicitudes de Ajax que se realicen. En `xajax.callback.global.onResponseDelay` asignamos la función que se debe ejecutar al inicio de la solicitud y en `xajax.callback.global.onComplete` asignamos la función a ejecutar cuando finalice.

Procesar formularios múltiples con Xajax 0.5

En este caso, vamos a implementar un sistema de envío de formularios con estas diferencias:

- ✓ Vamos a utilizar la versión de Xajax 0.5 (antes estábamos utilizando la versión 0.2 y con la 0.5 varias cosas han cambiado)
- ✓ En este ejemplo vamos a realizar un tratamiento de caracteres UTF-8, para decodificarlos y utilizarlos en páginas que utilizan el juego de caracteres ISO-8859-1. Por supuesto, utilizamos el método de propio de Xajax 0.5.
- ✓ Por último vamos a introducir un procesamiento de múltiples formularios por página. En este ejemplo tendremos varios formularios en vez de uno y cada formulario servirá para actualizar un registro de una tabla de base de datos. Así permitiremos actualizar varios registros a través de la misma página.

Pues dicho esto, veamos primero la tabla de base de datos que vamos a actualizar por medio de los formularios.

	Campo	Tipo
<input type="checkbox"/>	<u>id_pais</u>	int(3)
<input type="checkbox"/>	nombre_pais	char(100)
<input type="checkbox"/>	bandera	char(30)

Es una tabla donde se guarda información de países. En concreto tenemos el identificador y luego el nombre del país y la bandera. Tanto la bandera como el nombre son cadenas de texto.

En nuestro ejemplo tendremos una página con todos los formularios para actualizar desde un mismo sitio todos los países metidos en la tabla. Se mostrará un formulario por país y cada formulario se procesará de manera independiente, alternando el país que se haya editado. Los resultados se cargarán en la misma página, informando si hubo errores de validación o si se actualizaron los datos correctamente.

Construcción de los formularios

Vamos a hacer un bucle para crear un formulario por registro de la tabla. Cada formulario tendrá un identificador distinto, para saber cuál es y además estará metido dentro de una capa DIV, que también tendrá un identificador propio. Para generar identificadores distintos para cada elemento utilizaremos una variable \$i para enumerar los formularios y las capas donde están.

Ahora podemos ver cómo se realiza un recorrido a todos los registros de la tabla, para crear un formulario por cada uno de ellos.

```
<?
//conecto con la base de datos
//este script está sobre mi instalación local de apache + php + mysql
$conn = mysql_connect ("localhost", "root", "");
//selecciono la base de datos con la que trabajar
mysql_select_db("dw");

$sql = "select * from pais";
$rs = mysql_query($sql);
$i=0;
while ($fila=mysql_fetch_object($rs)){
    $i++;
    echo '<div id=f' . $i . '><form id=form' . $i . '>
        <input type="hidden" name="formulario origen" value="f' . $i . '">
        <input type="hidden" name="id_pais" value="' . $fila->id_pais . '">
        <input type="text" name="nombre_pais" maxlength="100" size=50 value="' . $fila-
        >nombre_pais . '">
```

```

        <br>
        <input type="text" name="bandera" size=30 maxlength="30" value="" . $fila->bandera
        . "">
        <br>
        <input
            type="button"
            value="guardar"
            onclick="xajax_procesar_formulario(xajax.getFormValues('form' . $i . '\'))">
        </form></div>
        <div id=errorf' . $i . '></div>';
        echo "</p>";
    }

    mysql_close($conn);
?>

```

Nos podemos fijar que en cada formulario se incorporan dos campos `hidden`, uno con el nombre de la capa `DIV` donde está formulario de origen y otro con el identificador de país al que pertenecen los datos de ese formulario.

También hay que llamar la atención sobre el botón de envío del formulario, que no existe. En lugar del `submit` colocamos un botón normal que llama a una función xajax, que será la que se encargue de procesar el formulario. A esa función xajax le pasamos por parámetro los valores del formulario que debe procesar, que obtenemos por medio de la llamada al método Javascript `xajax.getFormValues()`, pasando a su vez el nombre del formulario.

Todo tiene sus respectivos identificadores, para poder referirnos a los formularios, las capas `DIV` que los contienen y la capa donde mostraremos los posibles mensajes de error.

Función Xajax para procesar el formulario

Esta función validará el formulario. Si hay errores de validación los mostrará y si no los hay intentará actualizar el registro de la base de datos. Si la sentencia SQL para actualizarlo falla, también mostrará el error y si se ejecuta correctamente avisará que se ha cambiado el registro.

```

function procesar_formulario($form_entrada){
    $respuesta = new xajaxResponse();
    //valido los datos
    $errores = "";
    if ($form_entrada["nombre_pais"] == ""){
        $errores = "Escribe un nombre de país";
    }elseif($form_entrada["bandera"] == ""){
        $errores = "Escribe el nombre de una imagen en el campo bandera";
    }

    if ($errores != ""){
        //hubo errores de validación en el formulario
        //muestro un mensaje de error.
        $salida="<b style='color:red;'>ERROR:</b>" . $errores;
        $respuesta->Assign("error"
        $form_entrada["formulario origen"],"innerHTML",$salida);
    }else{
        //si no tiene errores de validación el formulario
        $conn = mysql_connect ("localhost", "root", "");
        mysql_select_db("dw");
        $ssql = "update pais set nombre_pais=' ' . $form_entrada["nombre_pais"] . "',
        bandera=' ' . $form_entrada["bandera"] . "' where id_pais=' ' . $form_entrada["id_pais"];
        if (mysql_query($ssql)){
            $salida="<b style='color:green;'>OK!</b>";
            $respuesta->Assign($form_entrada["formulario origen"],"innerHTML",$salida);
            $respuesta->Assign("error"
            $form_entrada["formulario origen"],"innerHTML","");
        }else{
            $salida="<b style='color:red;'>ERROR</b>";
            $respuesta->Assign("error"
            $form_entrada["formulario origen"],"innerHTML",$salida . " <span style='font-size:8pt;'>"
            . mysql_error() . "</span>");
        }
    }
}

```

```

        mysql_close($conn);
    }
    //tenemos que devolver la instanciación del objeto xajaxResponse
    return $respuesta;
}

```

La función recibe por parámetro los valores del formulario que se ha enviado, en un array asociativo. El código está comentado para que se entiendan los distintos pasos. Ahora bien, lo importante es que se vea que dependiendo de lo que ocurra en la función se actualizan unos elementos de la página, para mostrar posibles errores o para decir que todo ha sido correcto.

Tratamiento UTF-8 en Xajax 0.5

Los datos que nos vienen a través de Ajax llegan al servidor con el juego de caracteres UTF8. En Xajax existen unos modos de convertir esos caracteres a ISO, si es que estamos utilizando la codificación ISO-8859-1. En Xajax 0.5 el tratamiento de los datos en UTF-8 cambia con respecto a la versión 0.2. Ahora debemos indicar que trabajamos con ISO-8859-1, después de instanciar el objeto de la clase Xajax y además tenemos que especificar una variable de configuración para decodificar automáticamente los caracteres UTF-8.

```

$xajax = new xajax();
$xajax->setCharEncoding('ISO-8859-1');
$xajax->configure('decodeUTF8Input',true);

```

Todo el código junto

Os dejo el script completo, para que podáis seguirlo con mayor facilidad y de manera global:

```

<?
//incluimos la clase ajax
require ('../xajax/xajax/xajax core/xajax.inc.php');

//instanciamos el objeto de la clase xajax
$xajax = new xajax();
$xajax->setCharEncoding('ISO-8859-1');
$xajax->configure('decodeUTF8Input',true);

function procesar_formulario($form_entrada){
    $respuesta = new xajaxResponse();
    //valido los datos
    $errores = "";
    if ($form_entrada["nombre_pais"] == ""){
        $errores = "Escribe un nombre de país";
    }elseif($form_entrada["bandera"] == ""){
        $errores = "Escribe el nombre de una imagen en el campo bandera";
    }

    if ($errores != ""){
        //hubo errores de validación en el formulario
        //muestro un mensaje de error.
        $salida="<b style='color:red;'>ERROR:</b>" . $errores;
        $respuesta->Assign("error" .
    $form_entrada["formulario_origen"],"innerHTML",$salida);
    }else{
        //si no tiene errores de validación el formulario
        $conn = mysql_connect ("localhost", "root", "");
        mysql_select_db("dw");
        $ssql = "update pais set nombre_pais='" . $form_entrada["nombre_pais"] . "',
bandera='" . $form_entrada["bandera"] . "' where id pais=" . $form_entrada["id_pais"];
        if (mysql_query($ssql)){
            $salida="<b style='color:green;'>OK!</b>";
            $respuesta->Assign($form_entrada["formulario_origen"],"innerHTML",$salida);
            $respuesta->Assign("error" .
    $form_entrada["formulario_origen"],"innerHTML","");
        }else{
            $salida="<b style='color:red;'>ERROR</b>";
            $respuesta->Assign("error" .
    $form_entrada["formulario_origen"],"innerHTML",$salida . "<span style='font-size:8pt;'>"
    . mysql_error() . "</span>");
        }
    }
}

```

```

        mysql_close($conn);
    }
    //tenemos que devolver la instanciación del objeto xajaxResponse
    return $respuesta;
}
$xajax->register(XAJAX_FUNCTION, 'procesar_formulario');

//El objeto xajax tiene que procesar cualquier petición
$xajax->processRequest();
?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>
<head>
    <title>Listado de breves</title>
    <?
        $xajax->printJavascript("../xajax/xajax/");
    ?>
</head>

<body>
<?
//conecto con la base de datos
//este script está sobre mi instalación local de apache + php + mysql
$conn = mysql_connect ("localhost", "root", "");
//selecciono la base de datos con la que trabajar
mysql_select_db("dw");

$ssql = "select * from pais";
$rs = mysql_query($ssql);
$i=0;
while ($fila=mysql_fetch_object($rs)){
    $i++;
    echo '<div id=f' . $i . '><form id=form' . $i . '>
        <input type="hidden" name="formulario_origen" value="f' . $i . '">
        <input type="hidden" name="id pais" value="' . $fila->id pais . '">
        <input type="text" name="nombre pais" maxlength="100" size=50 value="' . $fila-
>nombre_pais . '">
        <br>
        <input type="text" name="bandera" size=30 maxlength="30" value="' . $fila->bandera
. '">
        <br>
        <input type="button" value="guardar"
onclick="xajax_procesar_formulario(xajax.getFormValues(\'form' . $i . '\'))">
        </form></div>
        <div id=errorf' . $i . '></div>';
    echo "</p>";
}

mysql_close($conn);
?>

</body>
</html>

```

Cambiar el estilo CSS de una página

Vamos a hacer una página web que se puede visualizar con varios estilos CSS distintos, donde el usuario puede elegir el que prefiere. Por medio de diversos enlaces, el visitante podrá seleccionar el aspecto que desea para la web, entonces se cargará una hoja de estilos CSS distinta y por lo tanto cambiará el aspecto de la página.

Xajax contiene una serie de funciones para incluir archivos CSS en la página, así como para eliminar declaraciones de estilos incluidas previamente.

Las funciones de Xajax que permiten incluir o quitar declaraciones de estilos pertenecen al objeto de la clase `xajaxResponse`, que instanciamos en las funciones PHP que se tienen que procesar por medio de Ajax.

`includeCSS('nuevo_estilo.css')`

El método `includeCSS` sirve para incluir un link con una declaración de estilos en el `HEAD` de la página. Recibe el archivo, o mejor dicho, la ruta del archivo CSS que se desea incluir.

`removeCSS('estilo_a_quitar.css')`

El método `removeCSS` sirve para eliminar un link a una declaración de estilos insertada anteriormente con `includeCSS()` de Xajax. Recibe la ruta del archivo CSS que se desea quitar.

`waitForCSS()`

El método `waitForCSS()` sirve para obligar al objeto `response` a esperar que finalice la carga de un CSS antes de continuar realizando acciones.

Con esas tres funciones realizamos todo el trabajo de incluir y eliminar estilos muy fácilmente, como podemos ver en el siguiente ejemplo realizado con Xajax versión 0.5.

El código del ejemplo es el siguiente:

```
<?
session_start();
if (!isset($_SESSION["estilo_css"]))
    $_SESSION["estilo_css"] = "css_gris.css";

//clase ajax
require ('xajax/xajax_core/xajax.inc.php');
//instanciamos xajax
$xajax = new xajax();

function cambiar_estilos($nuevo_estilo){
    $respuesta = new xajaxResponse();
    $respuesta->includeCSS($nuevo_estilo);
    $respuesta->waitForCSS();
    $respuesta->removeCSS($_SESSION["estilo_css"]);
    $_SESSION["estilo_css"] = $nuevo_estilo;

    return $respuesta;
}

$xajax->register(XAJAX_FUNCTION, 'cambiar_estilos');

//procesar peticiones
$xajax->processRequest();
?>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>
<head>
    <title>Cambiar hoja de estilos</title>
    <link rel="stylesheet" type="text/css" href="<?echo $_SESSION["estilo_css"];?>">
    <?
    //librerías xajax javascript
    $xajax->printJavascript("xajax/");
    ?>

</head>

<body>

<h1>Página que cambia los estilos</h1>

<a href="javascript: void(xajax_cambiar_estilos('css_gris.css'))">Estilo gris</a> |
<a href="javascript: void(xajax_cambiar_estilos('css_verde.css'))">Estilo verde</a> |
<a href="javascript: void(xajax_cambiar_estilos('css_rojo.css'))">Estilo rojo</a>
```

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

En nuestro script utilizamos variables de sesión para recordar el estilo que ha seleccionado el usuario, de modo que se pueda mostrar la página siempre con ese estilo durante toda la sesión o visita del usuario al sitio web. Si deseásemos que el sitio recordase el estilo seleccionado en las diferentes visitas del usuario, aunque estas estuvieran muy espaciadas en el tiempo, podríamos utilizar Cookies con PHP.

En el script hemos realizado en un primer paso en el que se inicializa la sesión PHP y se genera una variable de sesión con el estilo del usuario, si es que no existía ya.

Luego incluimos las librerías Xajax y generamos el objeto de la clase Xajax.

Luego definimos una función, `cambiar_estilos($nuevo_estilo)`, que es donde realizamos el trabajo con Xajax para alterar los estilos de la página. Esta función recibe el nombre del archivo CSS que se debe incluir y realiza los siguientes pasos:

Lo primero que hace es crear un objeto de la clase `xajaxResponse`. El siguiente paso es incluir el archivo CSS con la función mencionada anteriormente, `includeCSS($nuevo_estilo)`, que depende del objeto `xajaxResponse`. Más tarde, con `waitForCSS()` obligamos a esperar que se haya cargado el nuevo archivo CSS. Por último eliminamos el archivo CSS anterior con `removeCSS()`, indicando como parámetro el nombre de la declaración de estilos que teníamos en la variable de sesión, y actualizamos la variable de sesión para recordar el estilo actual que ha seleccionado el usuario.

Las demás tareas del código de la página ya las debemos conocer si hemos seguido el manual de programación Ajax con PHP. Sólo cabe señalar que los tres enlaces de abajo tienen las distintas llamadas a la función Xajax, enviando como parámetro el nombre del archivo de estilos que se desea visualizar.

Una prueba que podemos hacer es cambiar el estilo varias veces y luego actualizar la página con F5. Veremos que la página memoriza el último estilo que se había seleccionado, ya que se guardó en una variable de sesión.

Generar eventos a elementos

Parecerá raro, pero desde funciones PHP podemos asignar comportamientos a elementos de la página, dinámicamente y por medio de Ajax. Esta es una de las facilidades que nos aportan las librerías Xajax, que ciertamente son muy de agradecer, especialmente para los programadores de PHP, que tienen a su disposición de una manera fácil y rápida las capacidades que trae Ajax para las aplicaciones web.

En este artículo vamos a ver cómo utilizar unos métodos de la clase `xajaxResponse` para asignar eventos y manejadores de eventos a cualquier elemento de una página web, desde funciones PHP.

Métodos `addEvent()` y `addHandler()` de `xajaxResponse`

Existen un par de métodos para asignar eventos a elementos de la página. Cualquier elemento de la página es susceptible de recibir instrucciones que se deben ejecutar como respuesta a

eventos producidos por el usuario. A través de estas dos funciones, muy parecidas, podemos asignar eventos Javascript, de los que ya conocemos anteriormente.

Para asignar un evento necesitaremos tres datos. Primero el identificador (atributo `id`) del elemento de la página al que se desea asignar el evento. Luego el nombre del evento (`onclick`, `onsubmit`...) y por último la función o código javascript a ejecutar como respuesta al evento.

Dado que, tanto `addEvent()` como `addHandler()` son métodos de la clase `sajaxResponse`, tendrán que ejecutarse como llamadas por medio de Ajax a funciones PHP. Veamos un ejemplo de uso:

Haremos un botón HTML, que no hace nada, porque no tiene ningún código Javascript asociado al evento `onclick`. Luego, por medio de Ajax y Xajax, asignaremos una sentencia asociada al evento `onclick`, que lógicamente, se ejecutará cuando se haga clic en el botón.

Este es el botón: (cabe fijarse en el atributo `id`)

```
<input type=button value="pulsame" id="miboton">
```

Ahora vamos a tener un enlace, que hará una llamada xajax para asignar ese evento:

```
<a href="javascript:void(xajax_agrega_evento())">Agregar un evento al botón</a>
```

Y ahora vamos a ver la función PHP `agrega_evento()`, que será llamada por Ajax y asignará el evento:

```
function agrega_evento(){
    $respuesta = new xajaxResponse();
    $respuesta->addEvent("miboton","onclick","alert('hola mundo')");
    return $respuesta;
}
```

Vemos que se hace uso del mencionado método `addEvent()`, que recibe tres parámetros: el identificador del elemento al que queremos asignar el evento, el nombre del evento que se quiere asignar y las sentencias javascript que se han de ejecutar cuando se produzca el evento.

Una vez ejecutada esta función por medio de Ajax, el botón habrá ganado el evento `onclick`, de modo que, cuando se pulse, aparecerá una caja de alerta con el mensaje "hola mundo".

Ahora, cabría decir que `addEvent()` no se encuentra muy documentado en la documentación actual de xajax 0.5, pero funciona. Por su parte, sí está documentado el método `addHandler()`, que es similar, salvo que `addHandler()` como último parámetro hay que indicar el nombre de una función Javascript que será encargada de procesar el evento. Por ejemplo, tenemos esta función Javascript:

```
<script>
function mifuncion(){
    alert ("Este segundo alert se debe al addHandler");
}
</script>
```

Y ahora podríamos asignar un evento `onclick` de esta manera:

```
$respuesta->addHandler("miboton","onclick","mifuncion");
```

Como se puede ver, se ha colocado el nombre de la función en el tercer parámetro.

Veamos un código de un ejemplo completo que utiliza estos dos métodos para asignar eventos:

```
<?php
//clase xajax
require ('../xajax/xajax core/xajax.inc.php');
//instanciamos xajax
```

```

$xajax = new xajax();

function agrega evento(){
    $respuesta = new xajaxResponse();
    $respuesta->addEvent("miboton","onclick","alert('hola mundo')");
    $respuesta->addHandler("miboton","onclick","mifuncion");
    return $respuesta;
}

$xajax->register(XAJAX_FUNCTION, 'agrega evento');

//procesar peticiones
$xajax->processRequest();
?>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>
<head>
    <title>Generar manejadores de eventos desde Xajax - Con Ajax y PHP</title>
<script>
function mifuncion(){
    alert ("Este segundo alert se debe al addHandler");
}
</script>
<?php
//librerías xajax javascript
$xajax->printJavascript("../xajax/");
?>

</head>

<body>

<h1>Creo manejadores de eventos a partir de Ajax</h1>

Aquí tengo un botón que no hace nada (púlsalo para comprobarlo), porque no se ha
indicado ningún manejador de evento
<br>
<input type=button value="pulsame" id="miboton">

<a href="javascript:void(xajax agrega evento())">Agregar un evento al botón</a>

</body>
</html>

</body>
</html>

```

Otros detalles a comentar sobre la declaración de eventos con Xajax

Si lo deseamos, podemos asignar varias funciones al mismo evento de un elemento HTML, con lo que se ejecutarán las distintas funciones en el orden en el que han sido asignadas al evento. Además, existe el método `removeHandler()`, también de `xajaxResponse`, que sirve para eliminar eventos que hayan sido declarados con `addHandler()`. El método `removeHandler()` recibe los mismos parámetros del `addHandler()` que queremos anular.

Material adaptado de:

Xajax - Funciones PHP

<http://www.programacionweb.net/articulos/articulo/?num=500>

Aprenda Xajax en 10 minutos

http://jvelazqu.glo.org.mx/index.php?option=com_content&task=view&id=17&Itemid=33

XAJAX - desarrolloweb.com

<http://www.desarrolloweb.com/articulos/introduccion-a-xajax.html>

Anexo I - Aprenda XAJAX en 10 minutos

Usando xajax en un *script* PHP

xajax está diseñado para ser extremadamente fácil de implementar tanto en aplicaciones web existentes como en nuevos proyectos. Puede añadir el poder de xajax a casi cualquier script PHP en siete sencillos pasos:

1. Incluya la clase xajax:

```
require_once("xajax.inc.php");
```

2. Cree una instancia del objeto xajax:

```
$xajax = new xajax();
```

3. Registre los nombres de las funciones PHP que usted quiere llamar a través de xajax:

```
$xajax->registerFunction("myFunction");
```

4. Escriba las funciones PHP que ha registrado y use el objeto `xajaxResponse` para que ellas retornen comandos XML

```
function myFunction($arg)
{
    // Inicializar el objeto xajaxResponse
    $objResponse = new xajaxResponse();
    $objResponse->addAssign("ElementId", "innerHTML", $newContent);

    //return the XML response generated by the xajaxResponse object
    return $objResponse;
}
```

5. Antes de que su script envíe cualquier salida, deje que xajax se encargue de cualquier petición

```
$xajax->processRequests();
```

6. Entre los tags `<head></head>` dígame a xajax que genere el JavaScript necesario:

```
<?php $xajax->printJavascript(); ?>
```

7. Llame a la función desde un evento JavaScript o una función de su aplicación:

```
<div id="SomeElementId"></div>
<button onclick="xajax_myFunction(SomeArgument);">
```

Eso es todo, xajax se encargará de casi todo lo demás. Su mayor tarea es escribir las funciones PHP y devolver las respuestas XML hechas por xajax para ellas, lo cual se hace extremadamente fácil con la clase `xajaxResponse`

¿Cómo actualizo mi contenido asíncronamente?

Quizás, la mayor y única funcionalidad de Xajax está en la clase `xajaxResponse`. Las otras librerías Ajax requiere que usted escriba vuestra propia función de retorno en javascript para tratar las informaciones retornadas por una consulta asíncrona y para actualizar tu contenido. Por otro lado Xajax permite controlar todo el contenido que proviene de php. La clase `xajaxResponse` permite crear instrucciones XML de tus funciones hechas en php y retornar variables a la aplicación. El XML es analizado por el motor de mensajes de ajax y las instrucciones dirán a xajax como actualizar el contenido de tu aplicación. La clase `xajaxResponse` ofrece actualmente un gran número de comandos útiles, tales como:

- ✓ `Assign`, que pone el atributo que se ha especificado en un elemento de tu página
- ✓ `Append`, que permite integrar el atributo especificado al final de un elemento de tu página
- ✓ `Prepend`, que permite agregar el atributo especificado al comienzo de un elemento de tu página
- ✓ `Replace`, que busca y reemplaza las informaciones del atributo especificado de un elemento de tu página
- ✓ `Script`, que ejecuta el código javascript especificado
- ✓ `Alert` que muestra una caja de alerta con el mensaje especificado
- ✓ ...

Una sola respuesta XML puede contener múltiples comandos, que serán ejecutados en el orden que han sido agregados en la respuesta, Por ejemplo, si el usuario hace click en un botón de la aplicación, el evento `onclick` llama al paquete javascript, que llama a la función php. Este paquete envía una consulta asíncrona al servidor a través de la función `XMLHttpRequest` donde xajax llama a la función php. La función php hace una búsqueda en la base de datos, algunas manipulan los datos, o los serializa. Utilizaremos la clase `xajaxResponse` para generar una respuesta XML xajax que contiene múltiples comandos que serán enviados en respuesta al motor de mensajes xajax encargado de ejecutarlas.

```
$objResponse = new xajaxResponse();
$objResponse->addAssign("myInput1", "value", $DataFromDatabase);
$objResponse->addAssign("myInput1", "style.color", "red");
$objResponse->addAppend("myDiv1", "innerHTML", $DataFromDatabase2);
$objResponse->addPrepend("myDiv2", "innerHTML", $DataFromDatabase3);
$objResponse->addReplace("myDiv3", "innerHTML", "xajax",
"<strong>xajax</strong>");
$objResponse->addScript("var x = prompt(\"Enter Your Name\");");
return $objResponse;
```

El motor de mensajes xajax va a analizar el mensaje xML y ejecutará el mismo.

- ✓ El valor del elemento que posee el `id myInput1` va a ser asignado a las informaciones en `$DataFromDatabase`.
- ✓ El color del texto que posee el `id myInput1` va a ser cambiado a rojo.
- ✓ Las informaciones en `$DataFromDatabase2` van a ser agregadas al final de `innerHTML` del elemento con el `id myDiv1`
- ✓ Las informaciones en `$DataFromDatabase3` van a ser agregadas al comienzo de `innerHTML` del elemento con el `id myDiv2`
- ✓ Todas las coincidencias de "xajax" en el `innerHTML` del elemento con el `id myDiv3` van a ser remplazadas por "`xajax`"; poniendo todas las coincidencias de la palabra "xajax" en negritas.
- ✓ Una caja de diálogo se mostrara pidiendo al usuario su nombre y apellido y el valor se guardará en una variable javascript llamada `x`.

Todo esto será implementado del lado del servidor en la función php para la creación y el envío a ajax de la respuesta XML.