EL PLUGIN jqUploader

INTRODUCCIÓN

El plugin jqUploader viene a sumarse a la oferta de herramientas gratuitas que hay para incorporarle a una web un mecanismo de subida de ficheros.

El mercado está lleno de sistemas que pretenden mejorar las funcionalidades que ofrecen los campos de tipo **`file`** para la subida de archivos al servidor. Si bien con la llegada de HTML 5 estos campos presentaron una mejora incuestionable, al permitir el uso del atributo **`multiple`** para enviar varios ficheros usando un solo campo, aún así no están a la altura del concepto de muchos desarrollos web. Por esta razón, han proliferado las soluciones, tanto gratuitas como de pago. Con jqUploader he pretendido incorporar a este repertorio una herramienta flexible, potente, altamente configurable, de diseño elegante y, por supuesto, gratuita. Y creo que lo he conseguido de largo.

Está basado en jQuery (de ahí el nombre) y en Bootstrap. Al ser tecnologías estandarizadas es perfectamente implementable en cualquier desarrollo web.

En este manual vamos a conocer el plugin, y todo lo necesario para usarlo en nuestros sitios web.

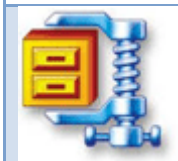
CARACTERÍSTICAS

Este plugin ofrece al desarrollador interesantes prestaciones que puede implementar en sus proyectos:

- Es altamente configurable, en cuanto a anchura y altura de la zona de subida de ficheros, colores, etc.
- Permite determinar qué tipos de archivos se podrán enviar.
- Permite establecer el peso máximo de cada archivo individual, y del conjunto total de archivos que se enviarán.
- Las validaciones de tipo de archivo y peso se realizan en el cliente, en tiempo real.
- Permite una previsualización en miniatura de las imágenes que se enviarán, si se ha configurado para admitir imágenes. Para otros tipos de archivos muestra unos iconos distintivos, que aparecen al final de esta sección.
- Permite que el propio plugin ofrezca al usuario un botón de subida de archivos, o que se use un botón, enlace u otro elemento de la página.
- Permite agregar archivos a la cola de subida mediante un botón específico, o mediante drag and drop sobre la zona de previsualización de los archivos.
- Una vez preparada una cola de archivos para subir, permite eliminar algunos individualmente antes del envío, o limpiar toda la cola de archivos.
- Permite añadir campos adicionales en la configuración, de modo que, junto a cada archivo, se mostrarán tales campos, y sus contenidos irán asociados a los archivos.
- Permite asociar al paquete de archivos otros campos de la página, de modo que, al hacer el envío, se manden todos los contenidos en un solo paquete.
- Incorpora una herramienta específica de PHP que permite gestionar los archivos enviados, los campos complementarios de cada uno, y los campos adicionales de la página, para grabar los archivos enviados en el servidor, y los datos en una base de datos, de forma cómoda y eficaz.
- Mensajes en múltiples idiomas. La versión actual (1.0) sólo incorpora inglés de Estados Unidos y español de España, pero, en breve, se añadirán otros idiomas.

ICONOS PARA REPRESENTAR LOS DISTINTOS TIPOS DE ARCHIVOS

Dado que esta herramienta permite subir, simultáneamente, distintos tipos de archivos, estos se representan, visualmente, mediante iconos (excepto las imágenes, que se representan mediante una miniatura de las mismas). Estos iconos son:

ICONO	TIPO DE ARCHIVOS ASOCIADO
	Archivos de audio
	Archivos de vídeo
	Documentos PDF
	Documentos de texto plano
	Comprimidos zip
	Otros tipos de archivo

ATENCIÓN. Dependiendo del navegador, y de la codificación de los archivos zip, estos no siempre son representados con su icono. En ocasiones se muestran con el icono de "Otros tipos de archivo". No obstante, sí son reconocidos y procesados correctamente.

DEPENDENCIAS

Cómo cualquier plugin actual, existen una serie de dependencias que debemos incorporar en nuestra página. Lo bueno de esto es que, debido a las tendencias actuales en diseño y desarrollo web, seguramente ya las tengas incorporadas, dado que forman parte esencial de casi cualquier trabajo:

DEPENDENCIAS EXTERNAS

Las dependencias que debes añadir a tu página, si aún no las tienes, son:

- Bootstrap
- jQuery
- jQueryUI
- La fuente Damion de Google Fonts [(1)]
- Los iconos de Font Awesome [(1)]

(1) Estas dos dependencias no tienes que añadirlas, ya que se encuentran incorporadas en el código del plugin. Por supuesto, si las necesitas para tu web, puedes añadirlas sin problemas.

DEPENDENCIAS INTERNAS

En tu página, además, deberás incluir el CSS y el JavaScript del plugin. Además, en el script que vaya a recibir los archivos, deberás incluír el PHP que se incluye en el paquete para el adecuado procesamiento. Por supuesto, si lo deseas, puedes procesar los archivos y el resto de campos enviados de forma totalmente manual (en este documento verás que es muy fácil hacerlo) pero la clase PHP que adjuntamos está pensada para facilitarte un poco las cosas.

LA PLANTILLA BÁSICA

La plantilla básica, con las necesarias dependencias, figura a continuación:

```
<!DOCTYPE html>
<html lang="en">
<head>
      <meta charset="UTF-8">
      <title>Ficheros</title>
      <!-- CSS de Bootstrap -->
      <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.mi
n.css">
      <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap-
theme.min.css">
      <!-- CSS de jqUploader -->
      <link rel="stylesheet" href="jqUploader/css/jqUploader.min.css">
</head>
```

```
<body>

	<!-- Código HTML 5 de la página -->

	<!-- El contenedor que serán convertido en la herramienta de
subida de archivos -->
	<div id="contenedor_de_archivos"></div>

	<!-- JavaScript de jQuery -->
	<script language="javascript" type="text/javascript"
src="https://code.jquery.com/jquery-3.2.1.min.js"></script>
	<!-- JavaScript de jQueryUI -->
	<script src="https://code.jquery.com/ui/1.12.1/jquery-
ui.min.js"></script>
	<!-- JavaScript de Bootstrap -->
	<script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.
js"></script>
	<!-- JavaScript de jqUploader -->
	<script language="javascript" type="text/javascript"
src="jqUploader/js/jqUploader.min.js"></script>

	<script language="javascript" type="text/javascript">
		/* Aquí se incluye el JavaScript del usuario, para la
		implementación del plugin y el resto del código de la
página. */
	</script>
</body>
</html>
```

USO BÁSICO

Vamos a empezar viendo cómo implementar jqUploader en nuestra página, de la forma más básica posible, a partir de la plantilla de la sección anterior. Lo que hacemos es referenciar el contenedor que hemos incluido para ser el mecanismo de subida de archivos, extendiéndolo con las funcionalidades del plugin, así:

```
var subidor = $('#contenedor_de_archivos').jqUploader();
```

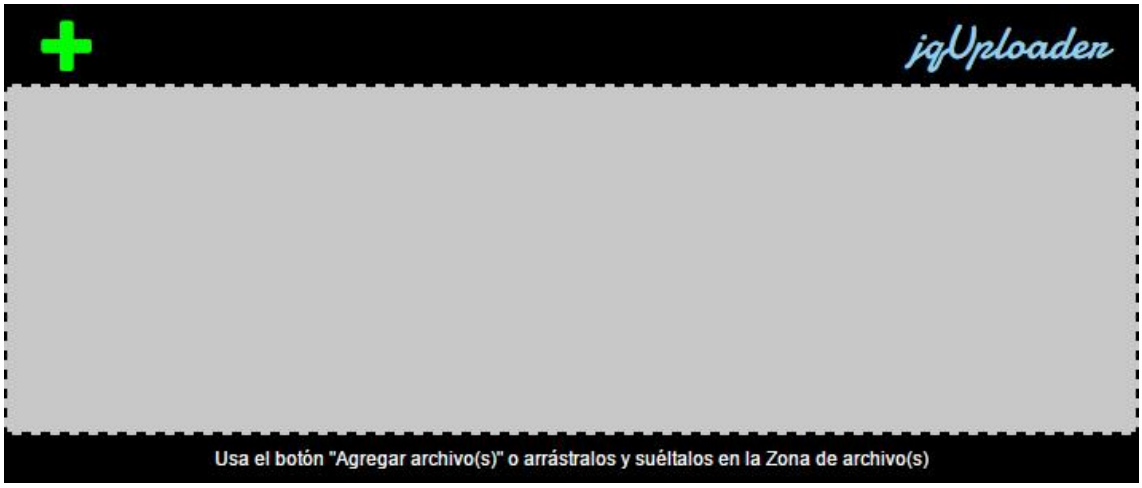
Por lo tanto, la plantilla nos quedará en una página, que podemos llamar test_simple.php, con el siguiente código:

```
<!DOCTYPE html>
<html lang="en">
<head>
	<meta charset="UTF-8">
	<title>Ficheros</title>
	<!-- CSS de Bootstrap -->
	<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.mi
n.css">
	<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap-
theme.min.css">
	<!-- CSS de jqUploader -->
	<link rel="stylesheet" href="jqUploader/css/jqUploader.min.css">
</head>
<body>
	<!-- El contenedor que serán convertido en la herramienta de
subida de archivos -->
	<div id="contenedor_de_archivos"></div>

	<!-- JavaScript de jQuery -->
	<script language="javascript" type="text/javascript"
src="https://code.jquery.com/jquery-3.2.1.min.js"></script>
	<!-- JavaScript de jQueryUI -->
	<script src="https://code.jquery.com/ui/1.12.1/jquery-
ui.min.js"></script>
	<!-- JavaScript de Bootstrap -->
	<script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.
js"></script>
	<!-- JavaScript de jqUploader -->
	<script language="javascript" type="text/javascript"
src="jqUploader/js/jqUploader.min.js"></script>

	<script language="javascript" type="text/javascript">
		/* Creamos un objeto para extender el contenedor con la
funcionalidad del plugin. */
		var subidor = $('#contenedor_de_archivos').jqUploader();
	</script>
</body>
</html>
```

Cuando ejecutamos la página en el servidor, vemos que el contenedor se ha convertido en un área de subida de archivos como el que aparece a continuación:



En la zona central, sombreada en gris y punteada alrededor, se alojará la cola de archivos que vayamos a enviar. Podemos añadir archivos arrastrándolos desde un directorio de nuestro equipo y soltándolos en dicha zona, o empleando el botón con el signo + que aparece en color verde, en la parte superior izquierda del subidor (botón **Agregar archivo(s)**).

Supongamos que vamos a enviar una imagen y un pdf. Tras seleccionarlos de cualquiera de las dos formas indicadas, el plugin tendrá un aspecto como el que ves a continuación:

Si alguno de los archivos no estuviera admitido por la configuración, no aparecería en la lista. También, si alguno de los archivos supera el peso máximo admitido se omitirá en la cola de archivos a enviar.

Ves que la imagen aparece miniaturizada, mientras que el pdf aparece como un icono que lo representa.

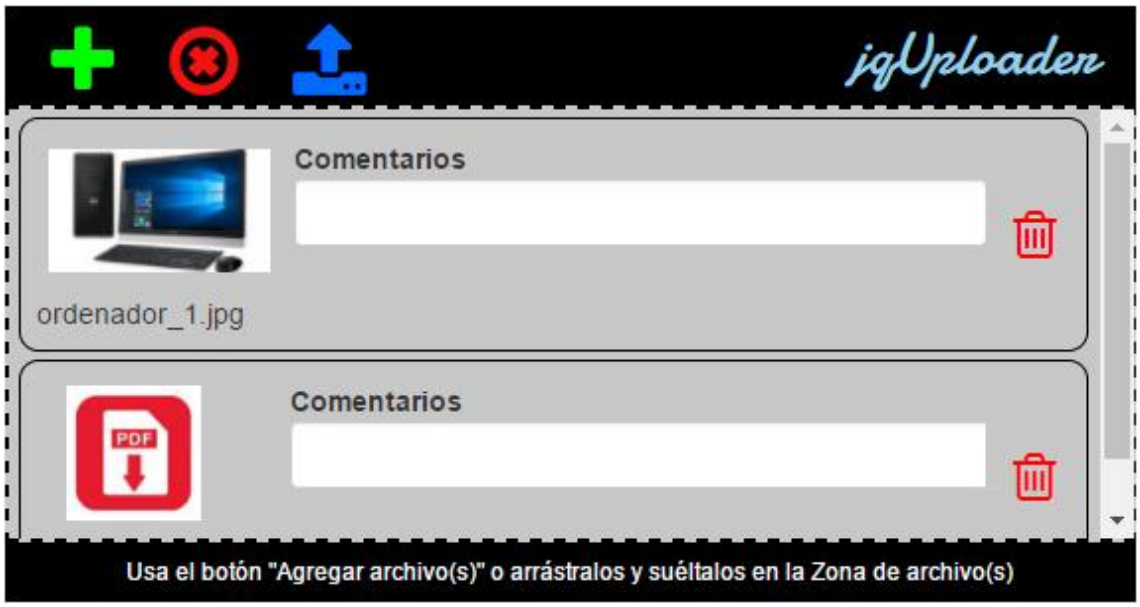
A la derecha aparece una barra de scroll, para que podamos desplazar la zona de la cola de archivos, y ver la parte que, por espacio, no es visible.

Debajo del icono aparece el nombre con el que tenemos el archivo en nuestro ordenador.

También ves, a la derecha de cada archivo, un icono en forma de papelera, que te permite eliminar cada archivo individualmente de la cola de archivos. Por ejemplo, si decides (antes de la subida) que el pdf no quieres enviarlo, puedes pulsar el icono de la papelera correspondiente y será eliminado de la cola de archivos.

La barra superior también ha cambiado. Ahora tienes un icono rojo, que te permite eliminar toda la cola de archivos (devolviendo el subidor a su estado inicial). También tienes un icono azul que te permite enviar la cola de archivos al servidor.

Supongamos que hubiéramos configurado nuestro plugin (más adelante veremos cómo), para añadir, a cada archivo, un campo de texto para que el usuario introduzca un comentario por cada archivo. En ese caso, el aspecto de la cola de archivos sería similar al siguiente:



CÓMO CONFIGURAR EL PLUGIN

Como suele suceder con los plugins jQuery, para configurarlo le pasamos las opciones en el momento de invocarlo. Por ejemplo, vamos a instanciar el plugin dándole una altura de 600 píxeles, poniéndolo en inglés, y estableciendo un color blanco para el área destinada a la cola de archivos. Lo haremos así:

```
var subidor = $('#contenedor_de_archivos').jqUploader({
      lang: 'en-US',
      color_fondo: 'rgb(255, 255, 255)',
      altura: '600px'
});
```

Como ves, las opciones se separan unas de otras mediante comas. El resultado será similar al siguiente:

A continuación, veremos la lista completa de opciones que podemos configurar.

LISTA DE OPCIONES

- **`url_destino`**. Nos permite indicar el script (y la ruta, en su caso) a donde se envían los archivos cuando se pulsa el botón de envío. Por defecto, el script se llama recibir.php y se encuentra en la misma ruta en la que alojamos el script desde el que importamos el plugin. Si vamos a especificar una ruta, esta será relativa a aquella en la que alojemos la página que hace uso del plugin, no a la ruta donde está dicho plugin. Más adelante, en este mismo documento, veremos cómo se reciben y procesan los archivos enviados. Un ejemplo de uso sería

```
url_destino: '../includes/grabar_ficheros.php'
```

- **`id_campo_file`**. El plugin usa, internamente, un campo file múltiple para agregar ficheros a la cola de envío. Por defecto, el nombre de este campo es **`id_file`**, pero podemos querer cambiarlo, si vamos a tener en la página un campo con ese id.

```
id_campo_file: 'mi_cola_de_ficheros'
```

`name_campo_file`. Se refiere al atributo **`name`** del campo de tipo file interno. Por defecto es **`name_file`**, pero podemos querer cambiarlo, por la misma razón que en el caso anterior.

```
name_campo_file: 'name_cola_de_ficheros'
```

- **`peso_maximo_de_cada_archivo`**. El plugin está pensado para permitirnos limitar el peso máximo de cada archivo, de forma que, si el usuario incluye un archivo de más peso, este no se añadirá a la cola de archivos a enviar, y se mostrará un mensaje de aviso en la parte inferior del subidor. El peso máximo de cada archivo está establecido, por defecto, en 2 Mb. Para establecer un peso máximo diferente, ponemos el valor numérico seguido, sin espacios, de una letra que será **K** (para Kb), **M** (para Mb) o **G** (para Gb). Por ejemplo, si queremos establecer un peso máximo por archivo de 15 Mb, lo haremos así:

```
peso_maximo_de_cada_archivo: '15M'
```

Si queremos que el usuario pueda enviar sus archivos sin limitación de peso, el valor a establecer es **0**, así:

```
peso_maximo_de_cada_archivo: '0'
```

- **`peso_maximo_de_la_subida`**. Es el peso máximo de toda la cola de archivos. Por defecto es de 10 Mb. Para establecerlo se emplean las mismas reglas que en el parámetro anterior: un valor seguido de **K**, **M** o **G**, o **0** si no queremos limitar el peso máximo de la cola de archivos. Ten en cuenta que, si tu sitio está construido con PHP, el archivo **php.ini** establece un valor **max_post_size**, que no puedes superar. Este parámetro no sobrescribe la configuración de PHP. Así, si tu PHP permite un máximo por envío de 64 Mb, hagas lo que hagas, no podrás superar ese peso máximo en la subida. Un ejemplo de uso sería:

```
peso_maximo_de_la_subida: '50M'
```

- **`minimo_numero_de_archivos`**. Se puede configurar el subidor para forzar al usuario a que suba un número mínimo de archivos, si nuestro diseño, o el uso que le vamos a dar a la aplicación, así lo requiere. Por defecto es 1 y, normalmente, no será necesario cambiarlo. Si tu aplicación requiere que el usuario suba, al menos, tres archivos en cada cola de envío, lo harías así:

```
minimo_numero_de_archivos: 3
```

- **`maximo_numero_de_archivos`**. Permite limitar el número máximo de archivos que se pueden incluir en una cola de envío. Por defecto, este parámetro tiene el valor 0, que significa que no estamos limitando la cantidad de archivos. Esta limitación sí es más habitual en una aplicación web. Un ejemplo de uso sería:

```
maximo_numero_de_archivos: 5
```

- **`lang`**. Establece el idioma. Por defecto es español de España (valor **'es-ES'**). Si queremos ponerlo en inglés de Estados Unidos, sería así:

```
lang: 'en-US'
```

Como hemos mencionado, estos dos son los únicos idiomas disponibles en la actualidad, aunque en el futuro se añadirán otros. Si establecemos un idioma que no está contemplado, se usará el idioma por defecto.

- **`mostrar_boton_de_subida`**. Este parámetro permite determinar si aparecerá o no el botón azul de subida de archivos, cuando haya una cola de envío. Por defecto, tiene el valor true. Se puede desactivar si el envío se va a hacer mediante un botón, enlace u otro elemento de la página ajeno al plugin. Lo desactivaríamos así:

```
mostrar_boton_de_subida: false
```

- **`boton_de_envio`**. Si el envío de la cola de archivos se va a hacer mediante un elemento externo al plugin, aquí estableceremos el valor id de dicho elemento. Por defecto, este parámetro tiene una cadena vacía, lo que indica que no se va a enviar mediante un elemento externo. Si, por ejemplo, el enlace que queremos usar para el envío tiene el valor envio en su atributo id, lo especificaremos así:

```
boton_de_envio: 'envio'
```

Lo que tenga dicho enlace en su atributo **href** es irrelevante, ya que el plugin anula su comportamiento por defecto. Sin embargo, por razones obvias, es importante que ese enlace no se use para ninguna otra cosa en la página.

- **`mostrar_boton_de_borrar_todos`**. permite indicar si aparecerá, en la parte superior del subidor, el botón rojo destinado a limpiar completamente la cola de envío. Por defecto, su valor es true. Si queremos que no aparezca, debemos usar:

```
mostrar_boton_de_borrar_todos: false
```

En ese caso, no se podrá limpiar toda la cola de envío de golpe, sino que, lo que se quiera eliminar, se deberá quitar archivo a archivo, mediante el icono de papelera que aparece a la derecha de cada uno.

- **`anchura`**. Permite establecer la anchura del plugin en la página. Por defecto está al 60% del ancho del viewport. Se puede establecer en porcentaje, o en una medida absoluta en píxeles. Por ejemplo, si queremos que la anchura sea siempre de 800 píxeles, lo haremos así:

```
anchura: '800px'
```

- **`anchura_min`**. Permite establecer la anchura mínima del plugin. Esto tiene sentido cuando al parámetro anterior se le ha dado un valor porcentual. Por defecto, está en 500 píxeles, pero podemos cambiarlo (tanto con un valor fijo como porcentual). Por ejemplo, para establecer que la anchura sea de, al menos, el 80% del viewport, lo haremos así:

```
anchura_min: '80%'
```

De todos modos, no es aconsejable establecer este parámetro en porcentual, ni por debajo de los 500 píxeles.

- **`altura`**. Permite establecer la altura global del plugin. Por defecto es 300 píxeles. Aunque se puede establecer como porcentual, con las alturas puede ser conflictivo en algunos casos, como ya saben los usuarios de CSS. Un ejemplo de uso sería:

```
altura: '600px'
```

- **`altura_min`**. Permite establecer la altura mínima del plugin. Por defecto está en 300 píxeles, aunque podemos cambiar esto, así:

```
altura_min: '500px'
```

ATENCIÓN. Como es lógico, si la anchura es menor que la anchura mínima o la altura es menor que la altura mínima (lo que no tiene sentido que hagamos) tienen preferencia los valores mínimos.

- **`margen_sup`**, **`margen_inf`**, **`margen_izd`** y **`margen_der`**. Estos cuatro parámetros (que pueden establecerse independientemente), permiten fijar los márgenes alrededor del subidor. Por defecto están fijados en 10 píxeles por arriba y por abajo, y en auto por los lados, para que quede centrado horizontalmente en la página.

- **`color_fondo`**. Permite establecer el color de la zona principal, destinada a la cola de envío. Por defecto está en un tono gris medio. El color se puede establecer en **hexadecimal**, en **rgb** o en **rgba**, del mismo modo que se hace en CSS. Por ejemplo, para poner la zona central en blanco, usaremos:

```
color_fondo: 'rgba(255, 255, 255, 1)'
```

o bien:

```
color_fondo: 'rgb(255, 255, 255)'
```

o también:

```
color_fondo: '#FFFFFF'
```

o:

```
color_fondo: '#FFF'
```

- **borde**. nos permite establecer el borde que rodea la zona central, destinada a la cola de envío. Por defecto, el valor es **'2px dashed black'**, pero podemos establecerlo siguiendo las normas de CSS.
- **accion_de_subida_correcta**. Cuando se envían archivos al servidor puede que haya algún problema de envío, recepción o procesamiento. En ese caso, se nos informará mediante un mensaje, y tendremos opción a reintentar el envío. Si todo ha ido bien, también saldrá un mensaje. En ese caso, pueden pasar dos cosas: que, simplemente, se limpie la cola de envío, sin afectar al resto de la página, o que se recargue toda la página completa. La primera es la opción por defecto, con el valor **'C'**. Si queremos que, tras un envío correcto, se recargue la página entera, lo estableceremos así:

```
accion_de_subida_correcta: 'R'
```

- **tipos_de_archivo**. Permite indicar que tipos de archivos se admitirán para ser enviados. Por ejemplo, podemos querer que nuestros usuarios puedan enviar documentos PDF y archivos de audio en mp4, pero ningún otro tipo de archivo. En ese caso, estableceremos este parámetro asi:

```
tipos_de_archivo: new Array(
     'application/pdf',
     'audio/mp4'
)
```

Los formatos de archivo admitidos por defecto son: **'application/pdf'**, **'application/zip'**, **'application/mp4'**, **'audio/mpeg'**, **'audio/mp4'**, **'audio/mp3'**, **'image/*'**, **'text/plain'** y **'video/mpeg'**.

- **campos_complementarios**. Podemos, como ya sabemos, hacer que a cada archivo se le asocien determinados campos complementarios, para añadir datos que luego querremos, por ejemplo, grabar en una base de datos relacionando cada campo con su correspondiente archivo. Por ejemplo, nuestra aplicación permite al usuario subir fotografías, agregándole, a cada una, un comentario. Por defecto, los archivos suben sin campos complementarios. Si queremos establecerlos, lo haremos así:

```
campos_complementarios: new Array(
     new Array('Comentarios', 'text', 'comentarios',
'maxlength=10')
     new Array('Detalles', 'text', 'detalles', 'maxlength=10')
),
```

Esto hará que a cada archivo se le añadan dos campos: uno para comentarios y otro para detalles, por ejemplo. Como ves, los campos que tendrán los archivos se definen en una matriz y cada elemento de la misma (cada campo) es, a su vez, una matriz con cuatro datos. Son (por el orden en que aparecen):

(1) La etiqueta con la que se mostrará el campo al lado de cada archivo.
(2) El tipo de campo. En esta versión del plugin sólo se admite, por ahora, **text**, **password**, **number** o **date**.
(3) El valor del atributo **id** y **name** de dicho campo. ***Deberemos evitar colisiones de nombres con otros campos que usemos en la página (esto es, desde luego, vital para que todo funcione bien).***
(4) Una cadena con los atributos que queremos añadirle a los campos. Por ejemplo, en el caso de campos de tipo **number**, esta cadena podría ser algo así: **'min=0, max=50, step=2'**.

- `campos_procedentes_de_la_pagina`. Esta opción (cuyo valor por defecto es un array sin elementos, le da una gran flexibilidad a este plugin, ya que permite referenciar campos de la página externos al propio plugin, pero que serán empaquetados y enviados al servidor cuando se pulse el botón de envío. Un uso muy común es agregar a los ficheros enviados datos complementarios que, al contrario de lo que hemos visto antes, no son propios de cada archivo enviado, sino comunes a todos. El valor que le pasamos es una matriz conteniendo los atributos id de los campos de la página que queremos que se empaqueten y envíen con los archivos de la cola de subida. Un ejemplo sería así:

```
campos_procedentes_de_la_pagina: new Array(
      "oculto_cadena_1",
      "oculto_cadena_2",
      "oculto_cadena_3"
)
```

EL ENVÍO

El envío puede contener (así está previsto) una serie de ficheros de diversos tipos, así como campos complementarios, y otros campos de la página.

Para posibilitar la correcta implantación de todas las funcionalidades del plugin, ha sido necesario recurrir al empleo de la clase **FileReader** lo que, a su vez, imposibilita el envío de los ficheros empaquetados en un objeto **FormData**. Por esta y otras razones, se envía todo el contenido por **POST**. El script receptor (trabajando con PHP) recibe todos los datos en **$_POST**, no haciendo uso de **$_FILES**. Los campos son enviados como tales y los archivos se envían codificados en base 64.

Todo el conjunto es empaquetado en una estructura JSON, por lo que los scripts que reciban y procesen el paquete de datos deben estar grabados en UTF-8, y trabajar en esta codificación. Esto, en sí, no es ningún problema, ya que esta es la codificación normalizada para cualquier proyecto actual.

La cadena JSON con todo el material pasa en una variable llamada **`cadenaDeDatos`**, por lo que no deberás emplear este nombre en tu script. Considéralo palabra reservada.

LA RECEPCIÓN

Está previsto recibir los datos en un script PHP. Las razones de haberlo orientado hacia este lenguaje son múltiples y obvias, por lo que no vamos a entrar en polémica al respecto.

Evidentemente, recibir un paquete conteniendo archivos codificados, campos complementarios relacionados con dichos archivos, y otros campos de la página original, todo codificado como una gran cadena JSON recibida por **$_POST** hace que, a priori, el material recibido resulte inmanejable. Por esta razón, hemos incluido una clase PHP, expresamente diseñada para poder manejar estos datos de forma simple, como si hubieran sido enviados del modo tradicional, empleando **$_FILES**.

La clase se llama **`RecoveryClass`** y se encuentra en el fichero **`jqUploader/php/RecoveryClass.php`**. Por supuesto, tú puedes reubicarla en tu proyecto donde te convenga. Sólo tenlo en cuenta a la hora de incluirla en el script receptor.

EL SCRIPT RECEPTOR

El script receptor, que como sabemos, por defecto se llama **`recibir.php`**, empezará con la inclusión de la clase:

```
include_once 'jqUploader/php/RecoveryClass.php';
```

Lo siguiente que hacemos es crear un objeto de esta clase, pasándole, como argumento, el valor de **`cadenaDeDatos`** que hemos recibido, así:

```
$objetoDeArchivos = new RecoveryClass($_POST["cadenaDeDatos"]);
```

A partir de aquí usaremos los métodos del objeto que hemos creado para manejar adecuadamente los archivos y los datos, según se refiere en la siguiente sección.

RETORNO

Para que el plugin nos informe correctamente de si se han grabado los ficheros en el disco, y los datos en la BBDD, debemos hacer que nos devuelva una matriz de elementos codificada en JSON. Esta matriz podrá tener los elementos que queramos, dependiendo de lo que sea que vayamos a hacer con ellos. Si bien, actualmente el plugin no incorpora funciones de callback para procesar los elementos de esa matriz, en un futuro próximo sí las incorporará.

Lo importante es que, tenga lo que tenga esa matriz, deberá incluir un elemento con la clave asociativa **`procesado`**. Esta contendrá el valor **`S`** si todo ha ido bien, o **`N`**, si se ha producido algún fallo. Al final de la siguiente sección vemos un ejemplo de cómo debe devolver los datos este script.

LA CLASE RecoveryClass

La clase nos puede devolver, según los datos que hayamos pasado al script receptor, hasta tres matrices:

- La de los ficheros que incluimos en la cola de envío de jqUploader
- La de los campos complementarios, si los hay, de cada uno de los ficheros
- La de los campos de la página, si los hay.

LA MATRIZ DE LOS FICHEROS

Para obtener la matriz de los ficheros usamos el método `getArchivos()` a partir del objeto que hemos creado en la sección anterior, así:

```
$matrizDeFicheros = $objetoDeArchivos->getArchivos();
```

Esta matriz tiene un elemento por cada uno de los archivos enviados. Cada elemento de un archivo tiene, a su vez, los siguientes elementos:

`name`. Es el nombre del archivo original (por ejemplo, foto.jpg, o documento.pdf).

`type`. Es el tipo MIME del archivo (por ejemplo image/jpeg o application/pdf).

`size`. Es el tamaño, en bytes, del archivo.

`file`. Es el archivo codificado en base 64.

`randomKey`. Es una clave que emplea el plugin y esta clase, para identificar internamente cada archivo. En seguida veremos para que sirve.

`nameToSave`. Es el nombre con el que se grabará el archivo en el disco, si no especificamos otra cosa. Más adelante, en esta sección, vemos cómo hacer la grabación del archivo en el disco del servidor.

LA MATRIZ DE DATOS COMPLEMENTARIOS

Si, al usar el plugin, establecimos que cada fichero debía tener uno o dos o n datos complementarios, podemos obtenerlos todos en una matriz mediante el método del objeto `getComplementarios()`, así:

```
$matrizDeDatosComplementarios = $objetoDeArchivos->getComplementarios();
```

La matriz de datos complementarios tiene un elemento por cada conjunto de los datos complementarios de cada uno de los archivos. Dicho elemento tiene, como clave, el mismo valor que tiene el fichero en `randomKey`. Por ejemplo, supongamos que hemos mandado dos archivos, y que, cada uno de ellos, tiene dos datos complementarios, a los que hemos llamado **comentarios** y **detalles**. Recuerda que esto se define como opción al implementar el plugin. El propio plugin le ha asignado a cada uno de los archivos una randomKey. Supongamos que el

primero tiene **ajjdjeejrt** y el segundo tiene **ieurktloma**. La matriz de datos complementarios tendrá la siguiente estructura:

```
["ajjdjeejrt"] => array (2)["comentarios"=>"Comentario del primer
fichero", "detalles"=>"Detalles del primer fichero"],
["ieurktloma"] => array (2)["comentarios"=>"Comentario del segundo
fichero", "detalles"=>"Detalles del segundo fichero"]
```

Por lo tanto, es extremadamente simple recorrer ambas matrices con un bucle **foreach**, para grabar en una base de datos cada uno de los campos asociado a su correspondiente fichero.

LA MATRIZ DE DATOS DE LA PÁGINA

Esta matriz es más simple aún de manipular. La obtenemos con el método `getCamposDePagina()` del objeto que estamos usando, así:

```
$matrizDeDatosDePagina = $objetoDeArchivos->getCamposDePagina ();
```

Cada elemento tiene como clave el nombre que tuviera el campo en nuestra página, y el correspondiente valor asociado.

GRABAR LOS FICHEROS EN EL SERVIDOR

Esto también es extremadamente fácil. El objeto que hemos creado tiene el método `saveFile()`, que recibe hasta tres argumentos, de los que sólo el primero es obligatorio:

- El elemento que queremos grabar. A partir de un bucle que recorra la matriz de archivos, podemos pasar el índice numérico del elemento en curso en cada iteración del bucle, o el valor del atributo name del archivo que estamos procesando en dicha iteración.
- La ruta en la que queremos grabarlo, con respecto a aquella en la que está el script actual. Si es una cadena vacía, se usará la ruta actual.
- El nombre con el que queremos grabar el archivo en disco. Si este es una cadena vacía se grabará con el valor de nameToSave del archivo. Si es un nombre, el método ya le añade la extensión que le corresponda, según el tipo de archivo, y lo graba con el nombre indicado.

EXCEPCIONES

El objeto de la clase RecoveryClass puede lanzar una excepción (que deberemos poder capturar), si al usar el método `fileSave()` nos referimos a un elemento que no existe en la matriz. En condiciones normales, recorriendo la matriz con un bucle, esto no va a suceder, evidentemente.

UN EJEMPLO DE SCRIPT DE RECEPCIÓN

A continuación, incluyo un ejemplo de script que recibe el paquete de datos, graba los ficheros en el disco, y graba en la base de datos los datos correspondientes, informando del resultado.

Para ello partimos de que hemos configurado el plugin con un campo complementario (que llamaremos **notas**) para cada archivo, y con dos campos adicionales de la página. El script de recepción podría ser algo como esto:

```
<?php
	/* Conexion con base de datos. */
	$conexion = new
PDO('mysql:host=localhost;dbname=base_de_datos;charset=UTF8', 'root',
'');
	$conexion->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);

	// Incluimos la clase RecoveryClass
	include_once 'jqUploader/php/RecoveryClass.php';

	// Creamos un objeto de la clase RecoveryClass
	$objetoDeArchivos = new RecoveryClass($_POST["cadenaDeDatos"]);

	// Recuperamos las tres matrices que han llegado por POST
	$matrizDeArchivos = $objetoDeArchivos->getArchivos();
	$matrizDeComplementarios = $objetoDeArchivos-
>getComplementarios();
	$matrizDeDatosDePagina = $objetoDeArchivos->getCamposDePagina();

	$fallo = false; // indicador de si ha habido fallo

	/*Grabamos los dos campos de la página en un registro de una
tabla llamada fichas.
	Leemos el id de la inserción, para luego asociar las notas de
todos los archivos a este envío. */
	try {
		$consulta = "INSERT INTO fichas (";
		$consulta .= "campo_1, ";
		$consulta .= "campo_2, ";
		$consulta .= "..., "; // Los campos que necesitemos, según
nuestra estructura de datos.
		$consulta .= "campo_n";
		$consulta .= ") VALUES (";
		$consulta .= "'".$matrizDeDatosDePagina['valor_1']."', ";
		$consulta .= "'".$matrizDeDatosDePagina['valor_1']."', ";
		$consulta .= "'".$matrizDeDatosDePagina['...']."', ";
		$consulta .= "'".$matrizDeDatosDePagina['valor_n']."'";
		$consulta .= ");";
		$conexion->query($consulta);
		$reg_insertado = $conexion->lastInsertId();
	} catch (Exception $e){
		$fallo = true;
	}

	if (!$fallo) { // Si, hasta ahora, todo va bien
		foreach ($matrizDeArchivos as $keyFile=>$file){
			/* Tratamos de grabar el archivo en curso en el
disco, en la ruta actual, y con el nombre almacenado en nameToSave */
			try {
				$objetoDeArchivos->saveFile($keyFile, '', '');
			} catch (Exception $e){
				$fallo = true;
				break;
			}
```

```
                  // Si no se ha producido excepción, grabamos el
campo notas del archivo en curso en la base de datos.
                  try {
                        $consulta = "INSERT INTO notas (";
                        $consulta .= "ficha, ";
                        $consulta .= "archivo_correspondiente, ";
                        $consulta .= "notas";
                        $consulta .= ") VALUES (";
                        $consulta .= $reg_insertado;
                        $consulta .= "'".$file['randomKey']."', ";
                        $consulta .=
"'".$matrizDeComplementarios[$file['randomKey']]['notas']."'";
                        $consulta .= ");";
                        $conexion->query($consulta);
                  } catch (Exception $e){
                        $fallo = true;
                        break;
                  }
            }
      }

      $resultado = array("procesado"=>($fallo)?"N":"S");
      $resultado = json_encode($resultado);

      echo $resultado;
?>
```