APJ

Framework de PHP Asíncrono y jQuery

Versión 1.8

Revisión enero de 2019

Ricardo Seiffert
seifox@hotmail.es

Contenido

INTRODUCCIÓN

¿Qué es APJ?

Es un Framework simple para PHP basado en el patrón MVC (Modelo-Vista-Controlador) con programación orientada al objeto (POO) que utiliza JQuery para la manipulación del DOM, enfocado en el intercambio de datos de forma asíncrona por medio de AJAX con formato JSON, entre el controlador y la vista. Su principal fortaleza es el desarrollo de aplicaciones web.

Cuenta con un ORM, que es un manejador de datos simplificado basado en PDO (Objeto de Datos de PHP). Esto reduce el uso complejo de sentencias SQL para manipular la base de datos.

¿Por qué Asíncrono?

Porque permite intercambiar información entre la *vista* (página HTML) y el *controlador* (programa del servidor PHP) sin tener que recargar la página y retornar valores y/o instrucciones desde el controlador para modificar la página (DOM) y manipularla casi a su antojo.

Todo aquel que ha hecho páginas o aplicaciones web, basadas en PHP u otro lenguaje del lado del servidor, sabe lo que significa el trabajo para repoblar un formulario, que ha sido enviado para guardar sus datos, encontrar errores de validación y dejar los campos como estaban, para que el usuario haga las correcciones pertinentes. Algunos pensarán, para eso está JavaScript, pero ¿qué pasa si la validación es contra información de una base de datos, o tener que hacer un formulario que cambie en forma dinámica dependiendo de ciertas condiciones, o hacer consultas interactivas que permitan desplegar capas de información que vienen de una base de datos, sin tener que refrescar la página completa?

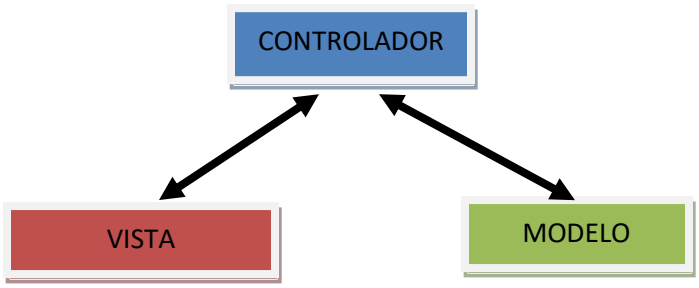
¿Por qué jQuery?

Porque es la librería de JavaScript más difundida y utilizada, con una comunidad muy grande. Se actualiza con cierta frecuencia y es difícil que desaparezca de la noche a la mañana, como podría pasar con otras similares.

¿Qué es MVC?

Modelo, Vista, Controlador es un patrón de arquitectura de software, que separa los datos, la lógica y la interfaz de usuario. Esto permite la reutilización y la separación de conceptos.

Representación gráfica:



En APJ, la comunicación entre la vista y el modelo no existe, todo pasa por el controlador. Por ello más que a un MVC, se puede parecer más a un MVP (Modelo Vista Presentador)

El Modelo: Es el que gestiona todo lo que tenga que ver con datos (base de datos). Acceso, grabado, validaciones y lógica del negocio.

La Vista: Es lo que vemos en pantalla y que interactúa con el usuario y el controlador. En nuestro caso la página HTML.

El Controlador: Es el que controla e interactúa entre la vista, el usuario y el modelo. En resumen, la lógica del programa.

¿Qué ventajas tiene?

Código reducido, reutilizable y ordenado, evitando la duplicidad de código, facilitando la mantención y/o expansión de los proyectos de desarrollo.

El manejo abstracto de cada capa facilita el entendimiento de un proyecto y permite trabajar mejor en grupos de desarrollo.

También evita el “código espagueti”, que es mesclar diferentes lenguajes en un mismo programa, como HTML, PHP, JavaScript y SQL, como lo habrán visto en algunas ocasiones.

REQUERIMIENTOS

- PHP 5.4 o mayor que tenga PDO habilitado

- MySQL con el motor INNODB habilitado

- Un navegador actualizado compatible con HTML5 (Chrome, Edge, Safari o Firefox)

- Conocimientos básicos de PHP y programación orientada a objetos

- Conocimientos básicos de HTML y CSS

- Conocimientos básicos de JQuery

- Conocimientos básicos de SQL

- Un editor de código o IDE, como pueden ser: *Aptana Studio*, *Atom, Brackets* y mi favorito *Nusphere PhpED* (pagado)

INSTALACIÓN

Por ahora la instalación es manual.

1. Bajar el framework APJ y copiar el contenido en la carpeta para su proyecto.
2. Deben estar las carpetas Libs, Models, Views, Helpers (opcional) y el archivo init.php
3. Editar el archivo init.php.
    - 3.1. Modificar la constante DEVELOPMENT para definir si se encuentra en modo de desarrollo o producción, por ahora dejar como TRUE, cuando lo suba a su sitio web, lo cambia a FALSE.
    - 3.2. Modificar la constante APPNAME por el nombre de su aplicación.
    - 3.3. Modificar la constante APP para definir la carpeta de su aplicación.
    - 3.4. Modificar la constante CONTROLLERS, normalmente igual a APP.
    - 3.5. Modificar la constante MODELS, define ubicación de los modelos.
    - 3.6. Modificar la constante MODELS_PREFIX, define el nombre con que deben identificarse los modelos.
    - 3.7. Modificar la constante MODELS_FILE_EXTENSION, define la extensión de los modelos.
    - 3.8. Modificar la constante HELPERS, define la ubicación de las clases adicionales o Helpers.
    - 3.9. Modificar la constante HELPERS_PREFIX, define el nombre con que deben identificarse los Helpers.
    - 3.10. Modificar la constante HELPERS_FILE_EXTENSION, define la extensión de los Helpers
    - 3.11. Modificar la constante VENDORS, define la ubicación de la carpeta que contiene las librerías de terceros,
    - 3.12. Modificar la variable *$domain* con el nombre del dominio que utilizara en producción. Deje *localhost* como está, es para desarrollo

3.13. Modificar $rootUrl para desarrollo y para producción
3.14. Modificar la constante LOGIN para definir el controlador de acceso a su aplicación.
3.15. Modificar la contante TIMEZONE para definir la zona horaria, si es que lo requiere.
3.16. Definir si la conexión a base de datos es persistente o no, con PERSISTENT_CONNECTION
3.17. Opcionalmente puede cambiar los formatos de visualización de la contante FORMAT
    3.17.1. *int*: Enteros (N° de decimales, separador de decimales, separador de miles)
    3.17.2. *decimal*: Decimales (N° de decimales, separador de decimales, separador de miles)
    3.17.3. *date*: Fecha (formato de PHP: d-m-Y)
    3.17.4. *datetime*: Fecha y hora (formato de PHP)
    3.17.5. time: Hora (formato de PHP: H:i:s)
    3.17.6. timestamp: Fecha y hora formato unix (formato de PHP)

4. Editar el archivo Libs/APJ/APJPDO.ini (archivo de configuración para el acceso a la base de datos)
    4.1. Cambie el valor de user, por la de un usuario válido, que tenga acceso a mostrar la estructura de las tablas (SHOW FULL COLUMNS FROM), No es obligatorio, pero en desarrollo es muy útil.
    4.2. Cambie el valor de password por una contraseña válida. Si la contraseña tiene símbolos como $, agregue comillas simples a la contraseña.
    4.3. Cambie el valor de dbname por en nombre de su base de datos.
    4.4. Opcionalmente puede especificar el charset.

Las carpetas de su aplicación

aplicación

- Libs: Carpeta contenedora de APJ y JQuery
    - APJ: Carpeta de la librería APJ
    - APJPDO_logs: Carpeta de los archivos de log de errores de base de datos
    - css: Carpeta de las css de los mensajes jAlert
    - images: Carpeta de las imágenes de los mensajes de alerta
    - js: Carpeta del módulo jqajaj.js
    - jQuery: Carpeta con la librería de jQuery
- Models: Carpeta de los modelos
- Views: Carpeta de las vistas
- Helpers: Carpeta de clases adicionales o helpers (opcional)
- css: Carpeta de las css para las vistas
- images: Carpeta de las imágenes usadas por las vistas
- Vendor: Carpeta de las librerías de terceros
- controladores.php: Cada controlador se almacena en la carpeta del proyecto
- init.php: archivo de constantes

El archivo init.php es requerido por cada controlador y define varias constantes para la aplicación.

Se debe cargar en cada controlador con *require("init.php");* o *require_once("init.php");*

Contenido de Libs/APJ

APJAutoload.php: Módulo auto cargador de clases (Controladores, Modelos y clases secundarias)

APJCommon.class.php: Métodos comunes de APJ.

APJController.class.php: Controlador padre.

APJHtmlGen.class.php: Generador de código HTML.

APJLog.class.php: Objeto que registra los errores generados por APJPDO o donde se solicite

APJModel.class.php: Modelo padre.

APJPDO.class.php: Objeto de control de base de datos, utilizado por los modelos.

APJSession.class.php: Objeto de control de sesiones de usuario.

jQ.class.php: Objeto que responde acciones jQuery desde PHP.

jQAction.class.php: Objeto que define los parámetros de las acciones del objeto jQ

jQSelector.class.php: Objeto que crea los selectores de jQ.

EL módulo jqajaj.js

Este módulo es el encargado de la comunicación por medio de Ajax, entre la vista y el controlador. Las principales funciones son APJSubmit() y APJCall(). Además contiene funciones que permite desplegar alertas formateadas, las cuales son: jInfo('información'), jWarning('advertencias'), jError('Errores'), jPrompt('solicitud de datos'), jConfirm('confirmar una acción').

jQuery

La carpeta contiene la librería de jQuery para uso local. Esta librería se puede ir actualizando con versiones más nuevas de jQuery. Ahora si lo prefiere, puede usar la librería directamente desde la web, conocidos como CDN: `src=https://code.jquery.com/jquery-1.12.4.min.js`

En lo personal prefiero trabajar con la copia local de jQuery, siempre será más rápido el acceso a la misma.

LA VISTA

Como se preparan las vistas para APJ. Primero debe saber que las vistas se almacenan en la carpeta Views o en la carpeta que haya configurado en la constante *VIEWS* de init.php

Ejemplo de la sección <head> de la vista (formato HTML5):

```
<head>
 <meta http-equiv="Content-Type" content="application/x-www-form-urlencoded; charset=UTF-8">
 <meta http-equiv="Lang" content="es">
 <title>APJ:{tituloVista()}</title>
 <script type="text/javascript" src="Libs/jQuery/jquery.min.js"></script>
 <script type="text/javascript" src="Libs/APJ/js/jqajaj.min.js"></script>
 <script type="text/javascript" src="Libs/jQuery/jquery.VerifyDateTime.min.js"></script>
 <script type="text/javascript">
        var url = "APJ:{self()}";
         var timeout = APJ:{timeout()};
 </script>
 <link rel="stylesheet" type="text/css" href="css/comun.css">
 <link rel="stylesheet" type="text/css" href="Libs/APJ/css/jqajaj.min.css">
```

Las líneas marcadas con negrita son requeridas en la sección <head> de cada vista que use un controlador para mostrarlas.

<script type="text/javascript" src="Libs/jQuery/jquery.min.js"></script>
Carga la librería de jQuery desde la ubicación local.

<script type="text/javascript" src="Libs/APJ/js/jqajaj.min.js"></script>
Carga el módulo jqajaj.js necesario para la comunicación con el controlador.

var url = "APJ:{self()}";
Define la variable *url*, que es el controlador con el cual se comunicará. La instrucción APJ:{self()} hace que el controlador sea el encargado de definir ese valor.

var timeout = APJ:{timeout()};
Define la variable *timeout*, qué es el tiempo de espera máximo en milisegundos para que el

servidor retorne una respuesta. Se puede definir manualmente o dejar al controlador que defina dicho valor, con APJ:{timeout()}

Opcionalmente, si queremos aplicar mayor compatibilidad HTML5 para navegadores como Internet Explorer o Firefox, que tienen problemas con los <input type=”date”> para los campos fecha. Recomiendo incorporar el módulo de compatibilidad jquery.VerifyDateTime.min.js de la siguiente forma:
<script type="text/javascript" src="Libs/jQuery/jquery.VerifyDateTime.min.js"></script>

<link rel="stylesheet" type="text/css" href="Libs/APJ/css/jqajaj.min.css">

Es la css que define los etilos de los mensajes de jAlert que utiliza jqajaj.js o jqajaj.min.js. Estos estilos se pueden cambiar para que se acomoden a su gusto.

Instrucción de precarga APJ:{}

Esta instrucción permite ejecutar cualquier método público del controlador, para que haga un reemplazo de algo que se mostrará en la vista.

Se puede ubicar en cualquier lugar de la vista. Lo que se encuentra entre las llaves {} es el método que ejecutará el controlador para cambiarlo por el contenido que devuelva el método invocado.

Ejemplo:

```
<tbody id="grillaCiudades">
        APJ:{grillaCiudades(CL)}
</tbody>
```

En el ejemplo, el controlador al cargar la vista y reemplazará la instrucción APJ:{grillaCiudades(CL)} con lo que devuelva el método grillaCiudades(), con el argumento ‘CL’.

Esto es muy útil para precargar con datos una tabla, un formulario o cualquier parte de la vista antes de mostrarla. Como se puede ver en el ejemplo de la sección <head>, donde dice <title>APJ:{tituloVista()}</title>, esto cambiará el título de la página, por lo que devuelva el método *tituloVista()*

La función APJCall()

APJCall() puede ser llamado solo después de haber desplegada la vista, ya que la función se encuentra en jqajaj.js, que se carga al mostrar la vista.

Sintaxis:

APJCall(‘metodo_a_llamar’,argumento1,argumento2…);

Lo que hace es llamar directamente un método del controlador para que ejecute alguna acción, con argumentos opcionales.

Ejemplos:

1.- con onclick o cualquier otro evento:

<button type=”button” onclick=”**APJCall(‘mostrarDatos’, 10, ’Ascendente’);**”>MOSTRAR </button>

2.- con jQuery:

<button type=”button” id=”mostrar”>MOSTRAR </button>

….

```
<script type="text/javascript">
        $(document).ready(function() {
                $("#mostrar").click(function() {
                APJCall(‘mostrarDatos’, 10, ’Ascendente’]);
                });
        })
</script>
```

Los dos ejemplos harían lo mismo, puede elegir la que más le acomode.

Lo que haría, es que cada vez que presione el botón MOSTRAR, llamaría el método *mostrarDatos*() con los argumentos 10 y ‘Ascendente’.

A menos que se use la propiedad useParametersAsArray = false; los argumentos se convierten internamente a un arreglo de parámetros, para pasarlo al controlador. Por lo mismo, el método mostrarDatos() debe recibir los argumentos como un arreglo. Para más detalles, ver métodos de respuesta del controlador.

Si no desea utilizar argumentos, solo óbvielos: APJCall(‘llamarLogin’);

Es muy importante que los botones que utilizan llamadas APJCall y APJSubmit sean definidos como type=”button”, si no, harán doble llamadas por *onclick* y por *submit.*

La función APJSubmit();

Función que permite enviar el formulario especificado y definir que método se ejecutará en el controlador.

Sintaxis:

APJSubmit('formulario', 'metodo', argumento1, argumento2, …);

El nombre del formulario y el método a llamar son obligatorios, pero los argumentos son opcionales.

Ejemplos:

1.- Envía el formulario 'formul' al método *cambioSeleccion()*, con el valor seleccionado como argumento, cada vez que se cambie la selección.

```
<form name="formul" id="formul">
        <select onchange="APJSubmit('formul', 'cambioSeleccion', this.value);">
        <option value="1">Activo</option>
        <option value="0">Inactivo</option>
        </select>
        …
```

2.- Envía el formulario al método *guardar()*

```
        <button type="button" onclick="APJSubmit('formul','guardar');">GUARDAR</button>
```

Las funciones JAlert

JInfo(), JWarning(), jError(), jConfirm(), jPrompt() y jProcess()

Estas funcionalidades fueron creadas para reemplazar a las funciones alert(), confirm() y prompt() de JavaScript, que son muy básicas.

Están diseñadas para desplegar y solicitar información al usuario, de tal forma que, a éste le sea imposible ignorarlas, ya que sitúa un fondo semi trasparente, que evita que el usuario haga otra acción que no se responder al mensaje.

Se pueden llamar tanto desde la vista, como del controlador, para dar información, advertencias, errores, confirmaciones, petición de datos e información de procesos.

Cada mensaje puede ser formateado por código HTML como un
 para hacer saltos de línea, agregar negrita con y otros.

jInfo()

Despliega información al usuario y este debe confirmar, presionando el botón Aceptar.

Sintaxis:

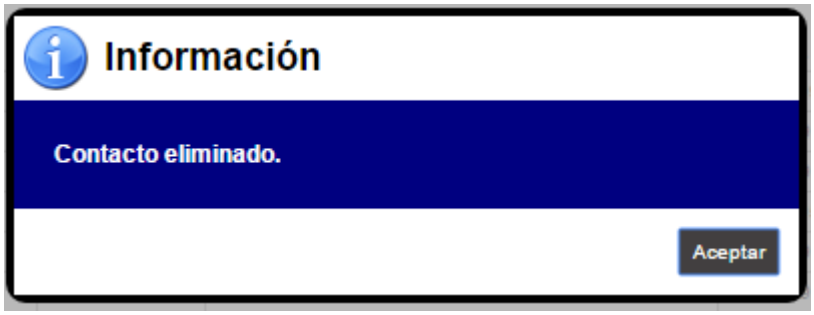
Desde la vista:
jInfo('Mensaje', 'Titulo', callBack, [arg1,arg2..]);

Desde el controlador:
Es la misma sintaxis, pero se le antepone $this->jInfo(...

El único argumento obligatorio es el mensaje, el resto es opcional.

Ejemplo:

jInfo("Contacto eliminado.");



Por defecto usará el título "información".

jInfo('Contacto eliminado.', 'Información', proceso, [1,'Si']);

Desplegará el mismo mensaje, pero además ejecutará la función javascript de retorno o CallBack llamada 'proceso' con los argumentos: 1 y 'Si', una vez que el usuario presione el botón Aceptar.

jWarning()

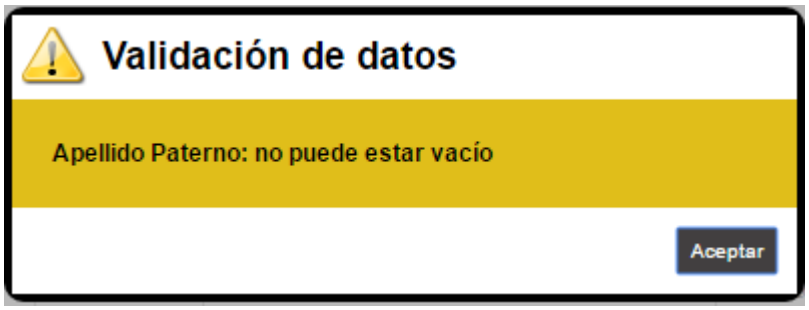
Similar a jInfo(). Se utiliza para lanzar advertencias

Sintaxis:

jWarning('Mensaje', 'titulo', callBack,[arg1,arg2…])

Ejemplo:

jWarning('Apellido Paterno: no puede estar vacio', 'Validación de datos');



JError()

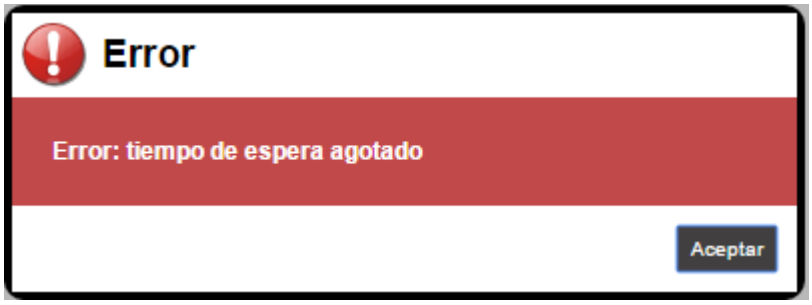
Similar a jInfo() y jWarning(). Diseñado para alertar de errores graves.

Sintaxis:

jError('Mensaje','titulo',callBack,[arg1,arg2…])

Ejemplo:

jError('Error: tiempo de espera agotado')



El título "Error" lo asigna por defecto.

jConfirm()

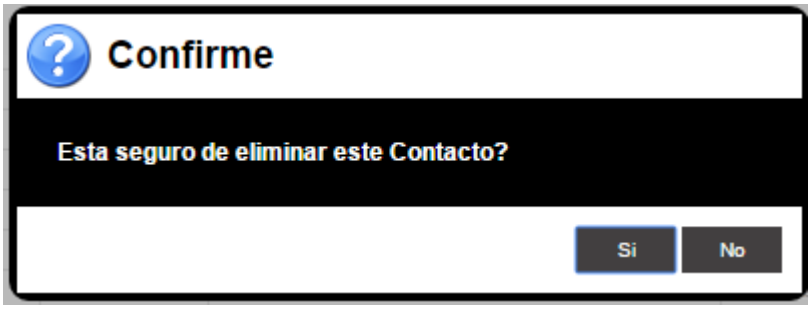
Está diseñado para confirmar acciones o dar respuestas de si o no. Solo si se responde "Si" ejecutará la función *callback* definida. Por ello, dicho parámetro es obligatorio. Los demás argumentos son opcionales.

Sintaxis:

APJConfirm('mensaje','titulo',funcionCallback,[arg1,arg2…]);

Ejemplo:

jConfirm('Esta seguro de eliminar este Contacto?', 'Confirme', APJCall, ['eliminar',101]);



Como pueden ver en el ejemplo se usó como función *callback* a la función APJCall, con los argumentos 'eliminar' y un segundo argumento que corresponde al 'Id' a eliminar.

Esto quiere decir que una vez que presione "Si", la respuesta será llamar la función de esta forma: APJCall('eliminar',101)

jPrompt()

Diseñado para la captura de algún valor. El valor introducido como respuesta pasará automáticamente como argumento de la función *callback* definida. Los argumentos adicionales son opcionales.

Sintaxis:

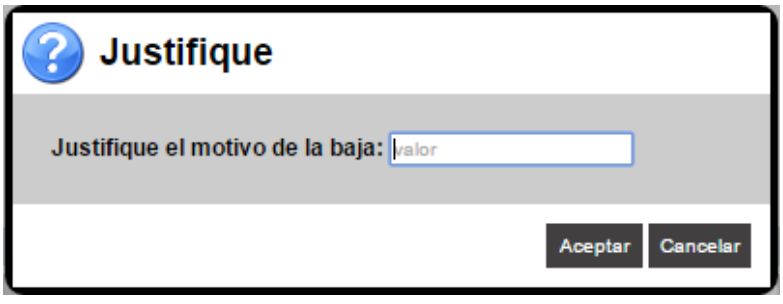
jPrompt('mensaje','titulo',callBack,[arg1,arg2]);

Ejemplo:

jPrompt("Justifique el motivo de la baja", "Justifique", darBaja);

darBaja será la función que ejecutará después de presionar "Aceptar" y el valor ingresado será el argumento.

```
function darBaja(motivo) {
   APJCall('darDeBaja', motivo);
}
```



jProcess()

Diseñada para mostrar que se están ejecutando procesos y que debe esperar a que terminen.

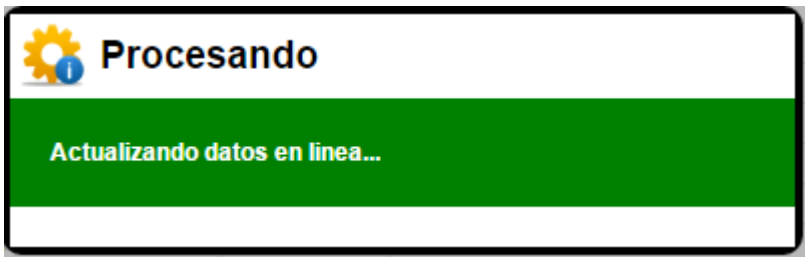
Sintaxis:

jProcess('mensaje', 'título', 'estilo')

El estilo es opcional y el único que acepta, por ahora, es 'blink', este parámetro define si quiere que el mensaje parpadee.

Ejemplo:

jProcess('Actualizando dato en línea…','Proceso en curso', 'blink')



La ventana de jProcess() se quita ejecutando otra función de *jAlert* o bien utilizando la función jClose()

jClose()

Función diseñada para forzar el cierre de cualquier ventana de jAlert. Cada vez que utilizamos cualquier función de jAlert, se ejecuta esta función para cerrar cualquier ventana abierta previamente. Se utiliza en especial para cerrar la ventana de jProcess().

EL CONTROLADOR (APJController)

El controlador es el corazón de cada módulo, controla tanto las vistas como los modelos. Se crean programando en PHP orientado a objetos.

Cada controlador hereda de la clase padre APJController, la cual contiene varios métodos y propiedades que nos ayudarán en la programación y el flujo de información entre la vista y los modelos, entre otras funcionalidades.

Sintaxis básica:

```
<?php
require_once("init.php");
class Consulta extends APJController
{
   private $modeloContactos;
   public function __construct($page) {
      $this->TimeOut = 20000;
      $this->sessionControl();
      $this->instanciaModelos();
      parent::__construct($page);
   }
   private function instanciaModelos() {
      $this->modeloContactos = new Model_Contactos();
   }
}
$app = new Consulta('consulta.html');
```

require_once("init.php");
Carga el archivo de constantes y el auto-cargador de clases.

class Consulta extends APJController{}
Se declara el controlador "Consulta".

private $modeloContactos;
Propiedad privada utilizada para instanciar el modelo de la tabla "contactos".
Se deben declarar una propiedad para cada modelo utilizado en el controlador.

public function __construct($page) {}
Es el método constructor, el que se ejecuta automáticamente al instanciar el controlador. Tiene como parámetro la página/vista a mostrar.

$this->TimeOut = 20000;
Aquí se está modificando la propiedad que define el tiempo de espera máximo, para ejecutar la comunicación entre la vista y el controlador. Esta propiedad es entregada a la vista por el método timeout(), como se vio en la sección <head> de la vista. Solo es necesaria modificarla si necesitamos cambiar el tiempo, ya que por defecto tiene un valor de 10000 (10 segundos).

$this->sessionControl();
Si queremos controlar el acceso a nuestra aplicación, este método es muy útil, ya que controla que se haya iniciado sesión con alguna cuenta de usuario, si ese usuario tiene acceso al controlador que estamos utilizando e incluso redirigir al controlador de *LOGIN* (definido en init,php) si es que la sesión ha caducado.

$this->instanciaModelos(); (opcional, ya que no todos los controladores usan modelos)
Método llamado para instanciar los modelos. Puede llamarse de cualquier forma, aquí se le puso instanciaModelos()

parent::__construct($page);
Instrucción que llama al método constructor de la clase padre. Es necesario si queremos que se muestre la vista automáticamente. Este método controla el flujo de la información, los comandos ejecutados desde la vista y viceversa, además de ejecutar el método *render()* que es el que muestra la vista.

$this->modeloContactos = new Model_Contactos();
Aquí se está instanciando el modelo "Contactos" y asignándose como objeto a la propiedad modeloContactos, para uso posterior.

$app = new Consulta('consulta.html');
Aquí se está instanciando el controlador "Consulta" y pasándole como argumento el nombre de la vista que debe desplegar.

Con estas instrucciones se ejecutará el controlador "Consulta", definirá el tiempo de espera en 20 segundos, controlará si hemos iniciado una sesión, instanciará el modelo "Contactos" y desplegará la vista que es la página "consulta.html".

Normas generales sobre los controladores

1. El archivo que contenga la clase del controlador debe llamarse igual que la clase, esto facilita la tarea al auto-cargador.
   Ejemplo: La clase MiControlador debería almacenarse en el archivo MiControlador.php

2. Todos los controladores deben heredar del controlador padre APJController.
   Ejemplo: Class MiControlador extends APJController

Propiedades del Controlador

Timeout: (entero) Tiempo de espera máximo de respuesta de la comunicación, en milisegundos.

userID: (entero) Identificador del usuario que ha iniciado sesión.

canRender: (booleano) Define si se puede mostrar la vista.

Form: (objeto) Objeto que captura automáticamente los datos del formulario, enviado con la función APJSubmit() desde la vista. Muy útil para el traspaso de datos a los modelos y otras acciones.

Si al ejecutar un APJSubmit(), se agregaron argumentos adicionales, estos serán agregados automáticamente al valor $this->Form->parameters, en formato de arreglo. $this->Form->parameters[0]..[1] etc.

useParametersAsArray: (booleano) Define si los parámetros pasados por APJCall son interpretados como un arreglo o como argumentos separados. Por defecto su valor es verdadero (true).

lastPage, currentPage, previousPage y nextPage: (entero) Propiedades para el manejo de la paginación de datos en formato de arreglo. Informan la última página, la página actual, la página anterior y la página siguiente (en el mismo orden).

Métodos del Controlador

render($page, $return=false): Despliega la vista *$page* fusionada con la precarga de los comandos APJ:{} desde la vista. El parámetro *$return*, por defecto es falso, le indica si debe desplegar directamente la página o debe devolverla como un valor.

Ejemplo:

```
$controlador = new Consulta(‘’); //instancia el controlador Consulta, con argumento en blanco

$pagina = $controlador->render(‘mipagina.html’, true); //Le solicita al controlador que le devuelva la página y la asigna a la variable $pagina, para uso posterior.
```

formToModel($model): Asigna los valores del formulario de la vista al modelo especificado. Solo pasará los valores de cuyos campos coincidan con los nombres de las columnas del modelo (tabla). Este método es muy útil ya que evita tener que pasar los valores uno a uno manualmente.

Ejemplo:

```
public function guardar() {
        $this->formToModel($this->modeloContactos); // Envía los datos del formulario al modelo
        If ($this->modeloContactos->guardar()) {  // Le solicita al modelo  el método guardar
                $this->jInfo(‘Los datos se han guardado’); //Informa que se guardaron
        }
}
```

formObjectToModel($model): Asigna los valores del objeto *Form* al modelo especificado. Solo pasará los valores de cuyos campos coincidan con los nombres de las columnas del modelo (tabla).

Este método es más útil que formToModel, ya que permite manipular los datos antes de enviarlos al modelo. Cabe recordar que el objeto Form se crea cada vez que se utilice APJSubmit() desde la vista.

Ejemplo:

```
public function guardar() {
  // El objeto Form es poblado con los datos del formulario por el llamado de APJSubmit()
  //Asigna un valor adicional al objeto Form antes de pasarlo al modelo.
  $this->Form->fecha_actualizacion = $this->currentDateTime();
    // Envía los datos del objeto Form  al modelo
    $this->formObjectToModel($this->modeloContactos);
    If ($this->modeloContactos->guardar()) {  // Le solicita al modelo  guardar los datos
      $this->jInfo(‘Los datos se han guardado’); //Si se guardaron, lo informa.
    }
}
```

modelToForm($model): Asigna los datos (una fila) leídos en el modelo y los asigna al objeto Form. Esto es muy utilizado para facilitar el manejo de los datos de una fila leída en el modelo. Es prácticamente el método contrario a *formObjectToModel()*

Ejemplo:

```
  public function editaContacto($params) {
    $id=$params[0];
    if ($this->modeloContactos->find($id)) { // Busca el contacto por ID, si lo encuentra:
      $this->modelToForm($this->modeloContactos); // Pasa los valores del modelo al objeto Form
      $this->fieldTypes['activo']="checkbox"; //Le define que 'activo' es un checkbox
      $this->setFormValues($this->Form); //Asigna los valores de Form al formulario de la vista
      $eliminar="<button type=\"button\" onclick=\"jConfirm('Esta seguro de eliminar este
Contacto?', 'Confirme', APJCall, ['eliminar',  {$this->Form->id}]);\"
title=\"Eliminar\">ELIMINAR</button>"; //Botón eliminar con sus instrucciones de confirmación
      $this->jQ("#eliminar")->html($eliminar); //Crea el botón eliminar en la vista
      $this->jQ("#form")->scrollTop(0); //Corre la visión  al tope del formulario en la vista
      $this->jQ("#dni")->focus(); //Ubica el cursor en el campo DNI
    }
  }
```

arrayToForm: Permite asignar los valores de un arreglo asociativo al objeto *Form*.

Ejemplo:

```
public function leeDatos($id) {
    $fila=$this->modeloContactos->buscarContacto($id); //Busca un contacto y lo devuelve en un
arreglo $fila
    $this->arrayToForm($fila); // Asigna los valores de $fila al objeto Form
    $this->actualizarFecha();
}

private function actualizarFecha() {
    $this->Form->fecha_modificado = $this->currentDateTime(); // Cambia fecha_modificado con
fecha y hora actual
    $this->formObjectToModel($this->modeloContactos); //Asigna los valores del objeto Form al
modelo Contactos
    $this->modeloContactos->update(); //Actualiza los datos de Contactos
}
```

clearForm(): Limpia o inicializa el objeto Form de valores.

Ejemplo:

```
private function limpiarForm() {
        $this->clearForm(); // $this->Form queda sin valores
}
```

objectToModel($object): Asigna los valores de un objeto/arreglo a las columnas del modelo

setFormValues($data, $form): Llena el formulario de la vista con los valores de un arreglo asociativo o los valores del objeto Form, si no ha especificado el parámetro *$data.*

$data: (opcional) Arreglo asociativo con los datos a pasar.
$form: (opcional) Nombre del formulario destino. Solo si desea usar los nombres de los campos a cambio de sus Ids.

Pasará los valores del arreglo u objeto Form, solo aquellos datos que coincidan con los '**id**' de los campos del formulario. Así no es necesario especificar a qué formulario debe pasarlos, ya que los Id's son únicos. O bien puede usar los nombres de los campos, para ello debe especificar el nombre del formulario por medio del argumento $form.

Este método también utiliza la propiedad *fieldTypes* para ayudar a poblar el formulario con aquellos campos más complejos de asignar valores/estados, como los *checkbox* o los *radio*. El resto de los campos no es necesario definir *FieldTypes*, ya que los obtiene desde el modelo.

Ejemplo:

```
prívate function poblarFormulario($datos) {
        $this->fieldTypes['activo'] = 'checkbox'; //define que el campo 'activo' es un checkbox
        If (is_array($datos)) { // si $datos es un arreglo
                $this->setFormValues($datos); // Llena el formulario de la vista con $datos
        } elseif ($this->Form) { //Por el contrario si el objeto Form tiene los datos
                $this->setFormValues(); // Llena el formulario con los datos del objeto Form.
        }
}
```

getForm(): Método que se ejecuta automáticamente cada vez que envía una solicitud APJSubmit(). Lo que hace es asignar al objeto *Form* los valores del formulario especificado por APJSubmit().

sessionControl(): Controla la sesión del usuario. Si la sesión no se ha iniciado, será dirigido al controlador de inicio de sesión, definido en la constante ***LOGIN*** de init.php. Por el contrario, si el usuario ya inició sesión y esta no ha caducado, la renovará y asignará la propiedad '*id*' del controlador, correspondiente al número de identificación único a cada usuario.

Bastará con ejecutarla una sola vez, por ejemplo, en el constructor del controlador y solo si es necesario controlar sesiones de usuario.

format($value,$type): Permite formatear valores según estructura de formatos dados en init.php. (También disponible en el modelo)

Los tipos de formatos permitidos son:

Decimales:
float, real (coma flotante de precisión simple),
double (coma flotante de doble precisión),
decimal (decimal fijo)

Enteros:
smallint, mediumint, integer, int, bigint, bit

Boléanos:
Boolean, bool, tinyint

Fecha y hora:
Date, datetime, time, timestamp

Ejemplo:

```
$valorFormateado = $this->format(1.45,'decimal');

// Devolverá 1,45 (Depende la consntante FORMAT de init.php)
```

convertDateTime($dateTime, $format): Convierte una fecha y/u hora según el formato dado. Si no se especifica el parámetro $format, asume el formato por defecto 'Y-m-d H:i:s'.
(También disponible en el modelo)

Ejemplo:

```
$fecha = "Lunes 5 de Abril de 2017";
$fechaFormateada = $this->convertDateTime($fecha, 'd/m/Y');

//Devolverá  05/04/2017
```

arrayInString($array, $string, $delim): Verifica si elementos de un arreglo se encuentran en una cadena, delimitado por el parámetro $delim, cuyo valor por defecto es ' ' (espacio). Devuelve True o False (También disponible en el modelo)

Ejemplo:

```
$colores=array(“rojo”,”blanco”,”azul”,”negro”);
$texto=”El gato de color negro se cruzó”;
If ($this->arrayInString($colores, $texto, “ “)) {
   echo “Uno de los colores está en el texto”;
} else {
   echo “Ninguno de los colores está en el texto”;
}
```

options($array, $valueIndex, $textIndex, $selected): Método que permite rellenar las opciones de un elemento <select> de la vista.
Parámetros:
$array: arreglo asociativo de valores para poblar los <option>
$valueIndex: Nombre de la columna que contiene los valores de *value* de cada <option>
$textIndex: Nombre de la columna que contiene las descripciones de cada <option>
$selected: (opcional) Elemento que se seleccionará por defecto, definido por el valor

Ejemplo:

```
//Solicitado por APJ:{optPaises()} desde la vista

  public function optPaises() {

   $paises = $this->modeloPaises->all('pais'); //Obtiene todos los paises del modelo Paises, ordenado por el nombre del país y lo asigna al arreglo $paises

   return $this->options($paises,'codigo','pais','CL'); /* Asigna el arreglo con los datos, el nombre de la columna clave, el nombre de la columna que contiene las descripciones y el elemento que debe quedar seleccionado por defecto (‘CL’)
Devolvería:
<option value=”AR”>Argentina</option>
<option value=”CL” selected>Chile</option>
etc.. */
  }
```

getLocalContent($url): Método para leer el contenido de un archivo local, definido en el parámetro $url. Si existe y es accesible, lo devolverá como una cadena (string), si no devolverá NULL.

Ejemplo:

```
public function muestraCondiciones() {
   if ($contenido = $this->leerArchivo('/anexos/condiciones.txt')) { //Solicita leer el archivo
      echo $contenido; // Lo muestra
   }
}
private function LeerArchivo($archivo) {
   return $this->getLocalContent($archivo); //Si existe y es legible lo retorna, si no devolverá NULL
}
```

getContent($url): Es muy similar al método getLocalContent, pero permite obtener el contenido de un archivo remoto como una página web, si el servidor remoto lo permite.

Ejemplo:

```
echo $this->getContent('http://www.dominio.com/pagina.html);
```

currentDateTime($format): Retorna la fecha y hora actual con el formato dado.
(También disponible en el modelo)

Ejemplo:

```
$this->Form->ultimaActualizacion = $this->currentDateTime('Y-m-d H:i:s');
//Asignará el valor de la fecha y hora actual, en formato Año-mes-día Hora:minuto:segundo
```

getStringBetween($string, $start, $end, $pos): Retorna una cadena ($string) que se encuentre entre los delimitadores ($start y $end), desde la posición inicial ($pos), cuyo valor por defecto es 0.
(También disponible en el modelo)

Ejemplo:

```
$cadena = "Esta es una cadena: Y esto es lo que debe devolver. El resto no interesa.
$textoResultante = $this->getStringBetween($cadena, ": ", ".");  // Retorna lo que se encuentra entre ": " y "." (Y esto es lo que debe devolver)
```

self(): Retorna el nombre del archivo que contiene el controlador actual. Este método es utilizado normalmente por la vista para asignar el valor de la variable *url*

Ejemplo:

```
//Controlador almacenado en index.php
public function obtenerArchivoControlador() {
   return $this->self(); //Retorna “index.php”
}
```

getController(): Retorna el nombre del controlador actual. Esta función puede ser útil para el control de sesiones, para saber si el usuario tiene acceso al controlador actual.

Ejemplo:

```
//Controlador index.php
public function obtenerControlador() {
   return $this->getController(); // Retorna “index”
}
```

paging($data, $limit=20,$page=1): Devuelve un arreglo con los resultados de una parte de un arreglo de datos. Esto es útil para crear paginaciones.

Argumentos:

$data: El arreglo con todos los datos
$limit: Define el número de elementos a devolver (por defecto 20)
$page: Página que desea que devuelva

Ejemplo:

```
public function paginación($pagina) {
   $paises=$this->modeloPaises->all(‘pais’); // Lee todos los países y los asigna a $paises
   $pagina=$this->paging($paises, 30, $pagina); // Retorna solo 30 países de la página definida, en este caso, el argumento $pagina
   return $pagina;
}
```

timeout(): Retorna el valor de la propiedad TimeOut, la cual es utilizada en la vista para definir el tiempo de espera máximo, en milisegundos, que debe esperar la vista, por una respuesta del controlador.

arrayToObject($array): Convierte un arreglo en un objeto.
(También disponible en el modelo)

Ejemplo:

```
$datos = array('Id'=>10,'Nombre'=>'Foo', 'precio'=>500);

$objeto = $this->arrayToObject($datos);
/* Devolverá un objeto del array y lo asigna a $objeto
$objeto->id será igual a 10
$objeto->Nombre será igual a 'Foo'
$objeto->precio será igual a 500
*/
```

redirect($url, $parent=false): Redirige o abre otro controlador, especificado por en $url. El parámetro $parent, que por defecto es false, define si debe redirigir para reemplazar el controlador actual (false) o el controlador padre/contenedor (true). Esta última opción es útil si por ejemplo el controlador se está llamando desde un <iframe>

Ejemplo:

```
// Si la sesión no está activa
if (!APJSession::active(APPNAME)) {
  $this->redirect('login.php',true); //Le solicita al controlador contenedor  que sea redirigido al controlador login.php
}
```

timeStamp(): Retorna la fecha actual en formato Unix
(También disponible en el modelo)

Ejemplo:

```
$this->modeloCliente->fecha_hora = $this->timeStamp();
```

setfieldType($field,$type='checkbox'): Asigna el tipo de datos que maneja el formulario. Este método se utiliza antes del método *setFormValues()* para indicarle como debe poblar los valores.

Solo es necesario utilizarlo si tenemos <input> del tipo "radio" o "checkbox" en el formulario de la vista. Tanto los radio como los checkbox deben tener el valor asignado. Ejemplo: value="1"

Ejemplo:

```
// Le define que el campo 'activo' es del tipo checkbox
$this->setFieldType('activo', 'checkbox');
//Le define que el campo 'genero' es del tipo opción (radio button)
$this-> setFieldType ('genero', 'radio');
$this->setFormValues($this->Form); // Asigna los valores del formulario de la forma correcta con los datos del objeto Form
```

Métodos de respuesta del controlador

Respuesta a una solicitud **APJCall()**

Ejemplo:

```
/* Ejemplo de una respuesta a una solicitud APJCall('solicitaSeccion', [$("#valor")->val(),'B'])
Aquí está solicitando al método solicitaSeccion () recibe 2 parámetros, el valor del objeto con id
"valor" y una constante 'B'
Los métodos llamados desde APJCall deben ser públicos */
public function solicitaSeccion($param) {
   list($valor,$seccion) = $param; // Esto equivale a $valor=$param[0]; $seccion=param[1];
   if ($fila = $this->modeloSeccion->find($valor)) { // Busca $valor en el modelo Seccion
      $this->jQ("#seccion")->val($fila[$seccion]; // Modifica el valor del objeto con id "seccion"
en la vista, con el valor dado
   }
}
```

Respuesta a una solicitud **APJSubmit()**

Ejemplo:

```
/* Ejemplo de respuesta a una solicitud APJSubmit('formulario1','guardar')
Aquí está enviando el formulario "formulario1" y solicitando el método "guardar()", sin
parámetros adicionales.
Los métodos llamados con APJSubmit deben ser públicos */
public function guardar() {
   $this->formToModel($this->modeloSeccion); //Pasa los valores del formulario al modelo
   if ($this->modeloSeccion->basicValidation()) { //Aplica la validación básica
      $this->showWarnings($this->modeloSeccion->errors,'Validación'); //Muestra los errores
      return false;
   }
   if ($this->modeloSeccion->guardar()!==false) { //Solicita al modelo guardar
      $this->jInfo("Datos guardados"); //Informa que los datos fuero guardados
      return true;
   }
   $this->showErrors($this->modeloSeccion->errors,'Error al guardar'); //Si hubo un error
}
```

Métodos jQ para jQuery

Los métodos jQ se pueden solicitar desde el controlador para manipular el DOM (Document Object Model)

Utiliza el mismo formato que jQuery, solo se cambia el . (punto) por el -> y el $ por $this->jQ

$this->jQ("#identificador")->val("1"); equivale en jQuery a $("#identificador).val("1");

Ejemplos:

```
public function creaTabla($tabla) {
   $this->jQ("#div2")->html($tabla); //Modifica el id="div2" por el contenido de $tabla
   $this->jQ(".tabla)->hide(); // Oculta todos los elementos que tengan la clase css "tabla"
   $this->limpiaFormulario('formulario1');
}

private function limpiaFormulario($formName) {
   $this->jQ('form[name="{$formName}"]')->resetForm(); //Inicializa el formulario1
   $this->jQ('#menos")->empty(); //Limpia el contenido del elemento con id="menos"
}
```

Para más detalle vea la documentación de jQuery API: https://api.jquery.com/

Métodos jAlert

Al igual que en la vista, desde el controlador se puede abrir cuadro del tipo jAlert

Ejemplo:

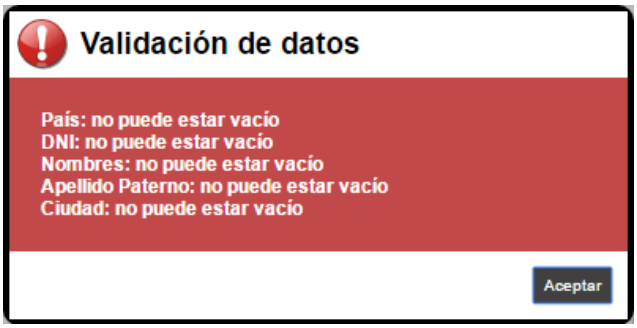
```
public function function solicitaConfirmacion($id) {
   // Solicita confirmar para eliminar el registro, utilizando como callback la función APJCall()
   $this->jConfirm("¿Confirme para eliminar este registro?", "Confirme", "APJCall",
['eliminar',$id]);
}

public function eliminar($params) {
   $id = $params[0]
   if ($this->modeloSeccion->delete($id)) {
      $this->jInfo("Registro eliminado");
   } else {
      $this->jError("No se pudo eliminar el registro");
   }
`
```

Para más detalles, vaya a la sección de funciones jAlert

Métodos para arreglos de Mensajes

showErrors($errors, $title, $alias): Despliega todos los mensajes de un arreglo, en un solo cuadro de mensaje con formato de la función jError().

Este método es muy útil para mostrar todos los errores o advertencias devueltos por algún modelo, al ejecutar alguna operación o validación.



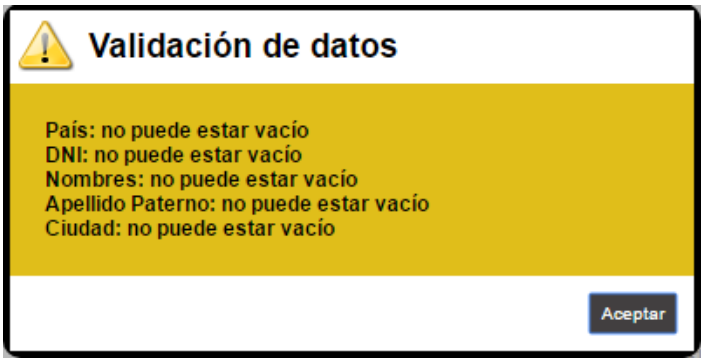
Argumentos:
$errors: array asociativo o numérico con los mensajes de error
$title: Cadena con el título del mensaje
$alias: (opcional) Un arreglo con los nombres de cada elemento

Ejemplo:

```
private function muestraErrores() {
   if ($this->modeloContactos->errors) {
      $this->showErrors($this->modeloContactos->errors,"Validación de datos",$this-
>modeloContactos->alias);
      return true;
   }
}
```

showWarnings($warnings, $title, $alias): Despliega todos los mensajes de un arreglo, en un solo cuadro de mensaje con formato de advertencia.

Este método es idéntico a showErrors, pero con formato de la función jWarning()



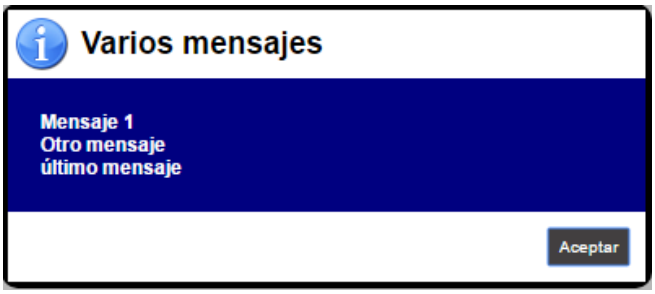
showMessages($messages, $title, $type): Muy parecido a los anteriores, solo que a cambio del parámetro alias, se define el tipo de formato de mensaje.

Argumentos:
$messages: Arreglo de mensajes
$title: Título del mensaje
$type: Tipo de mensaje, puede ser: 'Info', 'Error', 'Warning'

Ejemplo:

```
$mensajes=array("Mensaje 1","Otro mensaje","último mensaje");
$this->showMessages($mensajes, "Varios mensajes", "Info");
```

Muestra:



jScript($script): Método que permite ejecutar instrucciones JavaScript o jQuery

Ejemplo:

```
$this->jScript("$('#botonReset').clikc()"); Ejecutará instrucción jQuery para presionar el botón Reset
```

localizados por medio de su ID. Este método es muy útil para hacer listas desplegables de búsqueda y selección, justo debajo del elemento donde se introduce el texto buscado.

Ejemplo:

```
public function mostrarResultadoBusqueda($resultado) {
   $this->jShowDown("nombres", "contenedorDiv", $resultado);
}
```

Muestra:

Lo que hace jShowDown aquí, es ubicar la una lista desplegable (contenedorDiv) con los nombres coincidentes de la búsqueda parcial ($resultado), justo bajo el <input> de nombres. Cada fila de la lista debería tener asignado un APJCall(), que asigne el valor seleccionado definitivo.
Ver ejemplo01, controlador contactos.php, método buscarNombres() y en la vista contactos.html el botón que acompaña a nombres.

APJHtmlGen

Este objeto utilitario permite generar código HTML dentro del controlador para evitar devolver literales de código HTML a la vista. Lo que permite tener código más limpio dentro del controlador.

Instanciación del objeto:

Por variable local: $gen = new APJHtmlGen();

Por propiedad: $this->gen = new APJHtmlGen();

Métodos:

create($tag): Inicializa el contenido e inicia una etiqueta $tag HTML.

start(): Inicializa el contenido para comenzar a agregar elementos HTML

add($tag): Agrega o concatena una etiqueta $tag al contenido HTML

close(): cierra la última etiqueta abierta.

closeAll(): cierra todas las etiquetas abiertas.

preClose(): cierra parcialmente el último tag (>).

end($closeAll=true): cierra todas las etiquetas abiertas si $closeAll es true y devuelve el contenido HTML.

attr($attr, $value): asigna un atributo no definido en APJHtmlGen.

Ejemplo:

```
$this->gen->add('table')->clas('tabla')->add('tr')->add('td')->preClose()->add('textarea')->
attr('rows',5)->attr('cols',50)->text($texto)->closeAll();
```

clas($value): asigna una clase (class) a la etiqueta.

id($value): asigna el atributo id.

name($value): asigna la propiedad name.

value($value): asigna la propiedad value.

text($value): asigna el valor texto que va entre >Texto<

title($value): asigna la propiedad title.

type($value): asigna la propiedad type.

onclick($value): asigna la propiedad para el evento onclick.

onchange($value): asigna la propiedad para el evento onchange.

style($style): asigna la propiedad style.

Ejemplo:

```
private $gen;

 public function __construct($page) {
  $this->sessionControl();
  $this->instanciaModelos();
  $this->gen = new APJHtmlGen(); //Crea el objeto gen
  parent::__construct($page);
 }

public function grillaContactos($pais) {
  $this->gen->start(); //Inicia sin asignar ningún código
  if ($rows=$this->modeloContactos->consulta($pais)) {
   foreach ($rows as $row) {
    $rowf=$this->modeloContactos->setFormat($row);
    $this->gen->add("tr")->clas("modo1")->onclick("APJCall('editaContacto',{$row['id']})");
    foreach ($this->modeloContactos->fields as $campo) {
     $this->gen->add("td")->text($rowf[$campo])->close(); //Cierra el <td>
    }
    $this->gen->close(); //Cierra el <tr>
   }
  }
  return $this->gen->end(); //Devuelve las filas de la tabla
}
```

El Modelo (APJModel)

El modelo es el encargado del acceso a la base de datos, las validaciones de datos, las consultas y otras funcionalidades.

El modelo es una clase que hereda de la clase APJPDO, que es la que manipula la base de datos a través del objeto PDO. Por lo tanto, utiliza toda la funcionalidad de dicho objeto, como es el enlace de parámetros en las consultas (data binding). Esto nos permite hacer consulta SQL limpias, dinámicas y seguras, ya que previene la inyección de código maligno en nuestros datos.

Además, contiene funcionalidades de un ORM (mapeo objeto-relacional) básico. Esto facilita mucho las tareas básicas que se pueden ejecutar sobre la base de datos. En especial para los que no están muy familiarizados con el lenguaje SQL.

A pesar de que lo normal es que se asocie una tabla a cada modelo, no es estrictamente así, ya que podemos reasignar una tabla al modelo sobre la marcha, e incluso no asignarle ninguna tabla, o simplemente operar sobre varias tablas con sentencias SQL.

Sintaxis básica de un modelo:

```
class Model_Paises extends APJModel
{
  public function __construct() {
    parent::__construct();
    $this->setTable('paises');
  }
}
```

Todo nombre de una clase modelo debe comenzar con el nombre Model_, ya que de esa forma el método de auto-carga de clases detecta que es un modelo. Además cada modelo se debe guardar con la extensión ".model.php"

En el ejemplo el modelo "Paises" se llama Model_Paises, pero está almacenada en la carpeta "Models" con el nombre de archivo Paises.model.php. Es el auto-cargador de clases, el que se encarga de interpretar el nombre de la clase y cargar el archivo adecuado.

El pre-nombre Models, está definido en la constante MODELS del archivo init.php. Así qué, si quiere cambiar dicho nombre, lo puede hacer ahí, pero deberá seguir usando la extensión ".model.php"

Propiedades del Modelo.

table: contiene el nombre de la tabla de la base de datos, asociada del modelo. Esta propiedad se puede definir manualmente, en forma directa, solo si se define la estructura de forma fija. La otra forma es a través del método **setTable($table)** que, no solo define esta propiedad, sino que, además lee y define la estructura del modelo en forma dinámica cada vez que se instancia el modelo. Esto último, es muy útil cuando el proyecto se encuentra en etapa de desarrollo, ya que le evita estar redefiniendo la estructura del modelo, como pasa en otros FrameWorks.

Una vez que el proyecto se pasa a producción es recomendable definir la estructura del modelo de forma fija, lo cual aumenta el desempeño del modelo, ya que evita leer cada vez la estructura dinámicamente. Ver método showStructure().

pk: Es un arreglo que contiene el o los campos de la clave primaria. Esta propiedad se define automáticamente con el método setTable($table), o manualmente, en una definición fija.

structure: Arreglo que contiene toda la estructura de la tabla del modelo. Esta propiedad se define automáticamente con el método setTable($table), o manualmente, en una definición fija.

fields: Arreglo con los nombres de las columnas de la tabla asociada al modelo. Esta propiedad se define automáticamente con el método setTable($table), o manualmente, en una definición fija.

errors: Arreglo que contiene los errores o advertencias de una validación. Esta propiedad se puede manipular para definir errores adicionales que puedan surgir y así poder mostrarlo en el controlador.

alias: Arreglo de los nombres descriptivos de las columnas de la tabla asociada al modelo. Esta propiedad se puede asignar en forma manual o por medio del método setTable($table), el cual lee este dato de la propiedad “Comentarios” de la columna, en la estructura de la tabla en la base de datos. Ejemplo: nombre de la columna: fecha_nacimiento, Comentarios: Fecha de nacimiento

toLower: Array que contiene los nombres de columnas que se cambiarán a minúsculas antes de guardarlas en la tabla. Ejemplo: $this->toLower=array(‘email’,’codigo_postal’);

toUpper: Array que contiene los nombres de columnas que se cambiarán a mayúsculas antes de guardarlas en la tabla. Ejemplo: $this->toUpper=array(‘nombre’, ’apellido’, ’direccion’);

trim: Propiedad booleana que define si las consultas deben ser truncadas (quitar los espacios en blanco al comienzo y al final) antes ser utilizadas. Por defecto está definida como false.

trans: Booleana que permite saber si se ha activado el control de transacciones. Ver Control de Transacciones.

affected: Contiene el número de filas afectadas por la última consulta de acción (INSERT, REPLACE, UPDATE, DELETE, TRUNCATE)

numrows: Contiene el número de filas devueltas por una consulta de selección (SELECT)

error: Booleana que informa si ha ocurrido algún error al llamar algún método que ejecuta acciones sobre la base de datos.

errormsg: Cadena que contiene el último mensaje de error arrojado por alguna instrucción SQL que ha fallado.
Ejemplo:

```
if ($this->error) {
   return $this->errormsg; //Si hay un error, retornará el mensaje del error.
}
```

errornum: Entero que contiene el código de error arrojado por alguna instrucción SQL fallida.

lastPage, currentPage, previousPage y nextPage: (entero) propiedades para el manejo de la paginación de datos. Informan la última página, la página actual, la página anterior y la página siguiente (en el mismo orden).

Métodos del Modelo

setTable($table): Este método configura dinámicamente varias propiedades que definen la estructura del modelo para una tabla especifica. No es necesario utilizarlo cuando tenemos una definición fija del modelo. $table es el parámetro que define el nombre de la tabla que se asocia al modelo.

Ejemplo:

```
public function __construct() { //Constructor del modelo
   parent::__construct();
   $this->setTable('contactos'); // Define que la tabla del modelo se llama 'contactos'
   $this->trim=true; //Define que las consultas se truncarán de espacios en blanco
}
```

setAlias($names): Permite definir manualmente los "Alias" del modelo, si es que no se definieron en la estructura de la tabla, a través de la propiedad "Comentario".

Ejemplo:

```
private function defineAlias() {
   $alias=array('pais'=>'País', 'nombres'=>'Nombres', 'apellido_paterno'=>'Apellido Paterno');
   $this->setAlias($alias); //Define la propiedad 'alias' con el arreglo dado
}
```

showStructure(): Este método devuelve un <textarea> por pantalla que contiene todo el código necesario para definir el modelo en formato fijo, evitando leer la definición de la tabla en forma dinámica, cada vez que se utiliza. Puede ser utilizado tanto desde el controlador, como del mismo modelo, pero solo después de especificar la tabla con el método **setTable($table)**.

Se debe utilizar cada vez que se cambie la estructura de la tabla, para actualizar el código necesario para una definición fija.

Ejemplo:

```
// Desde el controlador
public function __construct($page) {
   $this->sessionControl();
   $this->instanciaModelos();
   parent::__construct($page);
}

private function instanciaModelos() {
   $this->modeloContactos = new Model_Contactos();
   $this->modeloContactos->showStructure();
}

// Desde el modelo
public function __construct() {
   parent::__construct();
   // Define la tabla del modelo (lee la estructura en forma dinámica)
   $this->setTable('contactos');
}
```

Al ejecutar el controlador del ejemplo, devolverá un cuadro de texto con un código como el siguiente:

```
$this->table = "contactos";
$this->structure =
array('id'=>array('Type'=>'int','Size'=>11,'Decimals'=>NULL,'Null'=>'NO','Key'=>'PRI','Default'=>
NULL,'Extra'=>'auto_increment','Comment'=>'Id'),'pais'=>array('Type'=>'varchar','Size'=>2,'Deci
mals'=>NULL,'Null'=>'NO','Key'=>'MUL','Default'=>NULL,'Extra'=>NULL,'Comment'=>'País'),'dni'
=>array('Type'=>'varchar','Size'=>25,'Decimals'=>NULL,'Null'=>'NO','Key'=>NULL,'Default'=>NU
LL,'Extra'=>NULL,'Comment'=>'DNI'),'nombres'=>array('Type'=>'varchar','Size'=>50,'Decimals'=
>NULL,'Null'=>'NO','Key'=>NULL,'Default'=>NULL,'Extra'=>NULL,'Comment'=>'Nombres'),'apelli
do_paterno'=>array('Type'=>'varchar','Size'=>40,'Decimals'=>NULL,'Null'=>'NO','Key'=>NULL,'D
efault'=>NULL,'Extra'=>NULL,'Comment'=>'Apellido
Paterno'),'apellido_materno'=>array('Type'=>'varchar','Size'=>40,'Decimals'=>NULL,'Null'=>'YES
','Key'=>NULL,'Default'=>NULL,'Extra'=>NULL,'Comment'=>'Apellido
Materno'),'ciudad'=>array('Type'=>'int','Size'=>11,'Decimals'=>NULL,'Null'=>'NO','Key'=>'MUL','
Default'=>NULL,'Extra'=>NULL,'Comment'=>'Ciudad'),'direccion'=>array('Type'=>'varchar','Size'
=>100,'Decimals'=>NULL,'Null'=>'YES','Key'=>NULL,'Default'=>NULL,'Extra'=>NULL,'Comment'=
>'Dirección'),'fono'=>array('Type'=>'varchar','Size'=>25,'Decimals'=>NULL,'Null'=>'YES','Key'=>N
ULL,'Default'=>NULL,'Extra'=>NULL,'Comment'=>'Fono'),'email'=>array('Type'=>'varchar','Size'=
```

Este código se copia completo y se pega en un método que puede llamar “defineEstructura()”, el cual debe llamar desde el constructor del modelo, para definir así, la estructura de forma fija. Como se especificó antes, esto es recomendable sólo si el modelo ya no cambiará y pasará a producción.

Después de pegar el código en un método que defina la estructura, debe recordar quitar el método **setTable($table)**, ya que no es necesario; lo está definiendo por código.

Como quedaría el modelo con estructura fija:

```
public function __construct() {
   parent::__construct();
   $this->defineEstructura();
}

private function defineEstructura() {
   $this->table = "contactos";
   $this->structure =
array('id'=>array('Type'=>'int','Size'=>11,'Decimals'=>NULL,'Null'=>'NO','Key'=>'PRI','Default'=>
NULL,'Extra'=>'auto_increment','Comment'=>'Id'),'pais'=>array('Type'=>'varchar','Size'=>2,'Dec
imals'=>NULL,'Null'=>'NO','Key'=>'MUL','Default'=>NULL,'Extra'=>NULL,'Comment'=>'País'),'dni
'=>array('Type'=>'varchar','Size'=>25,'Decimals'=>NULL,'Null'=>'NO','Key'=>NULL,'Default'=>NU
LL,'Extra'=>NULL,'Comment'=>'DNI'),'nombres'=>array('Type'=>'varchar','Size'=>50,'Decimals'=
>NULL,'Null'=>'NO','Key'=>NULL,'Default'=>NULL,'Extra'=>NULL,'Comment'=>'Nombres'),'apelli
do_paterno'=>array('Type'=>'varchar','Size'=>40,'Decimals'=>NULL,'Null'=>'NO','Key'=>NULL,'D
efault'=>NULL,'Extra'=>NULL,'Comment'=>'Apellido
Paterno'),'apellido_materno'=>array('Type'=>'varchar','Size'=>40,'Decimals'=>NULL,'Null'=>'YE
S','Key'=>NULL,'Default'=>NULL,'Extra'=>NULL,'Comment'=>'Apellido
Materno'),'ciudad'=>array('Type'=>'int','Size'=>11,'Decimals'=>NULL,'Null'=>'NO','Key'=>'MUL','
Default'=>NULL,'Extra'=>NULL,'Comment'=>'Ciudad'),'direccion'=>array('Type'=>'varchar','Size'
=>100,'Decimals'=>NULL,'Null'=>'YES','Key'=>NULL,'Default'=>NULL,'Extra'=>NULL,'Comment'=
>'Dirección'),'fono'=>array('Type'=>'varchar','Size'=>25,'Decimals'=>NULL,'Null'=>'YES','Key'=>N
ULL,'Default'=>NULL,'Extra'=>NULL,'Comment'=>'Fono'),'email'=>array('Type'=>'varchar','Size'=
>255,'Decimals'=>NULL,'Null'=>'YES','Key'=>NULL,'Default'=>NULL,'Extra'=>NULL,'Comment'=>'
email'),'fecha_nacimiento'=>array('Type'=>'date','Size'=>10,'Decimals'=>NULL,'Null'=>'YES','Ke
y'=>NULL,'Default'=>NULL,'Extra'=>NULL,'Comment'=>'Fecha
Nacimiento'),'activo'=>array('Type'=>'tinyint','Size'=>1,'Decimals'=>NULL,'Null'=>'NO','Key'=>N
ULL,'Default'=>0,'Extra'=>NULL,'Comment'=>'Activo'));
   $this->alias =
array('id'=>'Id','pais'=>'País','dni'=>'DNI','nombres'=>'Nombres','apellido_paterno'=>'Apellido
Paterno','apellido_materno'=>'Apellido
Materno','ciudad'=>'Ciudad','direccion'=>'Dirección','fono'=>'Fono','email'=>'email','fecha_naci
miento'=>'Fecha Nacimiento','activo'=>'Activo');
   $this->pk = array('id');
   $this->fields = array_keys($this->structure);
```

showModel($short=false): Este método es muy útil para documentar la estructura de la tabla asociada al modelo. El parámetro $short es un booleano que define si desea la documentación corta (true) o la estándar (false, por defecto).

Puede llamarse tanto desde el controlador como desde el modelo y devolverá la documentación en un cuadro de texto por pantalla.

Ejemplo:

```
// Desde el controlador
public function __construct($page) {
   $this->sessionControl();
   $this->instanciaModelos();
   parent::__construct($page);
}

private function instanciaModelos() {
   $this->modeloContactos = new Model_Contactos();
   $this->modeloContactos->showModel(false);
}
```

El ejemplo devolverá algo como lo siguiente:

```
/*
Table structure of [contactos]
+------------------+---------+------+----------+------+-----+---------+----------------+------------------+
|      Field       |  Type   | Size | Decimals | Null | Key | Default |     Extra      |     Comment      |
+------------------+---------+------+----------+------+-----+---------+----------------+------------------+
| id               | int     |   11 |          | NO   | PRI |         | auto_increment | Id               |
| pais             | varchar |    2 |          | NO   | MUL |         |                | País             |
| dni              | varchar |   25 |          | NO   |     |         |                | DNI              |
| nombres          | varchar |   50 |          | NO   |     |         |                | Nombres          |
| apellido_paterno | varchar |   40 |          | NO   |     |         |                | Apellido Paterno |
| apellido_materno | varchar |   40 |          | YES  |     |         |                | Apellido Materno |
| ciudad           | int     |   11 |          | NO   | MUL |         |                | Ciudad           |
| direccion        | varchar |  100 |          | YES  |     |         |                | Dirección        |
| fono             | varchar |   25 |          | YES  |     |         |                | Fono             |
| email            | varchar |  255 |          | YES  |     |         |                | email            |
| fecha_nacimiento | date    |   10 |          | YES  |     |         |                | Fecha Nacimiento |
| activo           | tinyint |    1 |          | NO   |     |       0 |                | Activo           |
+------------------+---------+------+----------+------+-----+---------+----------------+------------------+
*/
```

El texto desplegado lo podemos copiar, para pegarlo en la parte superior de la clase del modelo. Así tendremos documentado cada modelo y además nos servirá de referencia.

basicValidation(): Permite ejecutar una validación básica del modelo, antes de guardar, actualizar los datos. La validación se basa en los tipos de datos que tienen las columnas, si permite valores nulos, está definida la clave primaria, etc.

Devolverá un valor TRUE (verdadero) si **no** pasó la validación. Además se definirán las propiedades *errors* y *error*, para ser mostrado desde el controlador, con el método *showWarnings()* o *showErrors().* Por el contrario, si pasa la validación, devolverá un FALSE. Esto se pensó así, para evitar preguntar: if ($this->basicValidation() == FALSE) { o if(!$this->basicValidation()) {

Ejemplo desde el Controlador:

```
public function guardar() {
   $this->formToModel($this->modeloContactos); //Pasa los datos del formulario al modelo
   if ($this-modeloContactos->guardar()) { //Solicita al modelo guardar
      $this->limpiaFormulario(); //Si se guardó, limpia el formulario
   } else {
      $this->muestraErroresValidacion();
   }
}

private function muestraErroresValidacion() {
  if ($this->modeloContactos->errors) { //Hay mensajes de validación
     $this->showWarnings($this->modeloContactos->errors,"Validación de datos",$this-
>modeloContactos->alias); //Muestra los errores de validación con la descripción de cada
campo
   } else { // Si no, hay otro tipo de error
      $this->$this->jError($this->modeloContactos->errormsg);
   }
}
```

Desde el modelo:

```
public function guardar() {
   if ($this->basicValidation()) {
      return false; // No pasó la validación
   }
   if ($this->id) { // Si está editando, Id tendrá un valor
      return $this->update();
   } elseif ($this->insert()!==false) {
      return $this->lastId() //Retorna la última id insertada
   }
   return !$this->error; //Si no hay error, devolverá true
}
```

setFormat($row): Formatea los valores de las columnas, de la fila dada, según los tipos de datos definidos en el archivo init.php

Es muy útil para formatear campos fecha, entero, decimales, etc.

Ejemplo desde el controlador:

```
public function grillaContactos($pais) {
 $salida=" ";
 if ($rows=$this->modeloContactos->consulta($pais)) { //Consulta devuelta desde el modelo
  foreach ($rows as $row) { //Cada fila es procesada para crear la grilla de la tabla
   // setFormat() devuelve la fila formateada según los tipos de datos del modelo
   $rowf=$this->modeloContactos->setFormat($row); //$rowf tiene la fila con formato
   $salida.='<tr onclick="APJCall(\'editaContacto\',\'.$row['id'].')">';
   // Se crea cada columna de acuerdo a los campos del modelo
   foreach ($this->modeloContactos->fields as $campo) {
    $salida.="<td>".$rowf[$campo]."</td>"; //Crea la grilla con los valores formateados
   }
   $salida.="</tr>";
  }
 }
 return $salida; //Retorna el detalle de la grilla (<tbody>)
}
```

Las columnas del tipo *tinyint* las considera como booleanas y le aplicara el formato definido en init.php. Si no desea usar el formato asignado, utilice el valor original.

fomat($value, $type): Al idéntico al método del controlador, sirve para formatear valores, indicándole el tipo de valor que quiero que devuelva. Ver format() en la definición del controlador.

verifyDate($date, $format, $strict=true): Permite validar fechas, retorna false si no es válida y true, si lo es.

$date: Es la fecha a validar
$format: Es el formato de fecha a validar. Puede ser del tipo 'timestamp' o cualquier formato válido para PHP, como 'Y-m-d H:i:s'.
$strict: Define si se aplicará una validación estricta. (valor por defecto es true)

currentDateTime($format='Y-m-d H:i:s'): Devuelve la fecha y/o hora actual. El formato predefinido es Año-mes-día Hora:minutos:segundos.

clearValues(): Limpia los valores de las columnas del modelo.

server_info(): Devuelve la información del servidor de base datos.

Métodos ORM del Modelo

update($where='', $values=''): Ejecuta una instrucción de actualizar la tabla, según las condiciones de $where y con los valores de $values.

En la mayoría de los casos no es necesario utilizar los argumentos de los métodos ORM del modelo, si los datos ya han sido dados. Por defecto, update() verificara si está especificada la clave primaria, si es así, la utilizará para aplicar las condiciones de actualización.

Por otro lado, si deseamos especificar una condición, el argumento $where es muy dinámico, ya que se puede definir de muchas formas.

EL argumento dinámico **$where**

1. Si se pasa un número, asumirá que es la condición de la clave primaria (numérica):
   Ejemplo: $this->update(1);
2. Si se pasa un arreglo, asumirá que cada índice del arreglo es el nombre de la columna y cada valor es la condición, separado por un operador AND (Y).
   Ejemplo: $this->update(array('pais'=>'CL','ciudad'=>$ciudad,'activo'=>0), $valores);
   actualizará todas las filas que sean del país 'CL' y la ciudad se igual a la variable $ciudad y que activo sea igual a 0, con los valores del arreglo $valores.
3. Si se pasa un literal, lo entenderá como una condición normal de SQL
   Ejemplo: $this->update("país='CL' AND ciudad='{$ciudad}' AND activo=0", $valores);
   El mismo efecto que el anterior, solo que se utilizó una condición literal.

Este método devolverá el número de filas afectadas o false si hubo algún problema. El valor devuelto puede ser 0, si no hubo cambios que aplicar a la fila, así considere ello cuando le aplica alguna evaluación. If ($this->update() {...} puede dar un resultado negativo, ya que update() puede devolver 0, lo que dentro de la condición es entendido como falso. Es mejor consultarlo de esta otra forma: if($this->update()!==FALSE) {..} o simplemente consultar por $this->error, si es falso, no hubo problemas.

Ejemplo 1:

```
public function guardar() {
   if ($this->id) {
      $this->fecha_hora_actualizacion = $this->currentDateTime();
      $this->update();
   } else {
      $this->fecha_hora_creacion = $this->currentDateTime();
      $this->insert();
   }
}
```

El ejemplo muestra la forma más simple de actualizar o insertar una fila, ya que los datos fueron poblados desde el controlador por medio del método formToModel() o formObjetcToModel() y lo único que se modifica dentro del método guardar() es la fecha/hora de creación y actualización para cada caso.

Ejemplo 2:

```
public function guardar($valores) {
   if ($this->valores['id']) {
      $valores['fecha_hora_actualizacion'] = $this->currentDateTime();
      $this->update($valores['id'],$valores); //La condición y los valores son asignados
   } else {
      $valores['fecha_hora_creacion'] = $this->currentDateTime();
      $this->insert(false,$valores); //Se pasan los valores en arreglo
   }
}
```

Aquí los valores son pasados por parámetro en un arreglo asociativo.

Es importante destacar que, al asignar los valores del arreglo, estén bien asignados los índices, ya que deben coincidir con los nombres de las columnas del modelo, si no, desestimará los valores que no coincidan.

Formas de crear un arreglo asociativo:

1.- Con la función array:
$datos=array('id'=>1,'nombre'=>'Juan','fecha_nacimiento'=>'1990-12-30');

2.- Con asignación directa:
$datos['id']=1;
$datos['nombre']='Juan';
$datos['fecha_nacimiento']='1990-12-30';

insert($ignore=false, $values=''): Ejecuta una instrucción de insertar una nueva fila. El argumento $ignore, permite definir si la inserción ignorará los posibles errores por efecto de valores duplicados en claves primarias. Por defecto su valor es false. El argumento $values es opcional y corresponde a un arreglo asociativo de valores a insertar.

Este método devolverá el número de filas insertadas o false si hubo algún problema.

Ejemplo 1:

```
public function guardar() {
   if ($this->id) {
      $this->fecha_hora_actualizacion = $this->currentDateTime();
      $this->update();
   } else {
      $this->fecha_hora_creacion = $this->currentDateTime();
      $this->insert();
   }
}
```

Ejemplo 2:

```
public function guardar($valores) {
   if ($this->valores['id']) {
      $valores['fecha_hora_actualizacion'] = $this->currentDateTime();
      $this->update($valores['id'],$valores); //La condición y los valores son asignados
   } else {
      $valores['fecha_hora_creacion'] = $this->currentDateTime();
      $this->insert(false,$valores); //Se pasan los valores en arreglo lo que me obliga definir el
primer argumento de ignorancia de duplicados, en este caso, false.
   }
```

lastId(): Retorna la última Id numérica insertada en la clave primaria. Debe solicitarse después de insert() y antes de un commitTrans().

replace($values): Es igual al insert(), con la excepción de que si hay coincidencia con alguna clave primaria o índice único, la fila existente será eliminada y luego insertara la nueva. Se produce un reemplazo. No hay opción de ignorar los duplicados, ya que siempre los reemplazara.

Consideraciones a tener en cuenta al ser utilizado en claves primarias del tipo auto_incremental: Si no se entrega el valor del clave primaria, el replace() utilizará un nuevo valor. Por el contrario, si se da el valor de la clave primaria, la reutilizará.

El uso de replace() solo tiene sentido si la tabla tiene clave primaria y/o índice único

Ejemplo 1:

```
public function guardar() {
   if ($this->basicValidation()) {
      return false;
   }
   return $this->replace();
}
```

Aquí el método guardar() asume que los valores ya fueron asignados. Devolverá las filas afectadas por replace() o false si hubo algún error.

Ejemplo 2:

```
public function guardar($valores) {
      return $this->replace($valores); //$valores debe ser un arreglo asociativo
}
```

delete($where): Elimina el o los registros que cumplan la condición $where. Si no se especifica eliminará el registro que coincida con la clave primaria establecida.

Ejemplo1:

```
public function eliminar() {
   if ($this->id) {
      return $this->delete();
   }
   $this->errormsg=”No se ha seleccionado nada para eliminar”;
   return false;
}
```

Ejemplo2:

```
public function eliminar($pais,$ciudad) {
   $condicion=array(‘pais’=>$pais,’ciudad’=>ciudad);
   return $this->delete($condicion); //Eliminará todas aquellas filas que coincidan en el país y ciudad especificados
}
```

find($where): El método permite encontrar una fila del modelo, especificando condiciones, al igual que los métodos update() y delete, con una condición inteligente. (Ver el argumento dinámico **$where**).

find() devolverá como un array asociativo y al mismo tiempo asignando los valores al modelo, para tratar cada columna como parte del objeto.

Ejemplo1:

```
//Desde el controlador

public function buscar($id) {
   if ($fila=$this->modeloContactos->buscar($id)) {
      $this->muestraDatos($fila);
   } else {
      $this->jWarining(“Contacto no encontrado”);
   }
}

//Desde el modelo “contactos”

public function buscar($id) {
   return $this->find($id);
}
```

Ejemplo2:

```
//Desde el controlador

public function estaActivo($id) {
   if ($this->modeloContactos->verificaActivo($id)) {
      $this->jInfo("Está activo");
   } else {
      $this->jInfo("No está activo");
   }
}

//Desde el modelo "contactos"

public function verificaActivo($id) {
   $this->find($id);
   return $this->activo; //Como find() asocia la fila al modelo, puedo hacer referencia como si fuera una propiedad.
}
```

all($order): Devuelve todas las filas de todas las columnas del modelo en un arreglo asociativo. Equivale a un SELECT * FROM tabla ORDER BY $fila. El orden es opcional.

Ejemplo:

```
public function optionsPaises() {
   $paises = $this->modeloPaises->all('pais');
   return $this->options($paises,'codigo','pais','CL');
}
```

Like($field, $like): Ejecuta una consulta con criterios tipo LIKE, que le es aplicado a la columna *$field* y con el criterio *$like*. Devolverá todas las filas, en un arreglo asociativo, que cumplan con el criterio dado.

Ejemplo:

```
public function encontrarNombre($nombre) {
   $condicion = $nombre."%";
   return $this->like('nombre', $condicion); // Devolverá todas las filas, cuya columna nombre, comiencen con el valor de $nombre
}
```

avg($field, $condition): devuelve el promedio de la columna *$field*.

count($field, $condition): devuelve el número de elementos de la columna *$field*.

min($field, $condition): devuelve el valor mínimo de la columna *$field*.

max($field, $condition): devuelve el valor máximo de la columna *$field*.

sum($field, $condition): devuelve la suma de los valores de la columna *$field.*

Ejemplo:

```
public function promedio() {
   return $this->avg('edad'); // Devolverá el promedio de 'edad'
}
```

El argumento *$condition* es opcional y permite definir un criterio para las funciones descritas.

row($query, $params = NULL, $fetchMode = PDO::FETCHASSOC): Retorna una fila del modelo según la consulta SQL dada a *$query*, con parámetros de enlace opcionales y formato de retorno predefinido como arreglo asociativo (PDO::FETCH_ASSOC).

$query es una cadena que representa la consulta en el lenguaje SQL.
$params es un arreglo asociativo con los valores que se enlazan a la consulta.
$fetchMode es la forma que se desea que se devuelva la fila, la opciones son:
PDO::FETCH_ASSOC (defecto) devuelve un array indexado por los nombres de las columnas del conjunto de resultados.
PDO::FETCH_BOTH: devuelve un array indexado tanto por nombre de columna, como numéricamente con índice de base 0 tal como fue devuelto en el conjunto de resultados.
PDO::FETCH_CLASS: devuelve una nueva instancia de la clase solicitada, haciendo corresponder las columnas del conjunto de resultados con los nombres de las propiedades de la clase.
PDO::FETCH_NUM: devuelve un array indexado por el número de columna tal como fue devuelto en el conjunto de resultados, comenzando por la columna 0.
PDO::FETCH_OBJ: devuelve un objeto anónimo con nombres de propiedades que se corresponden a los nombres de las columnas devueltas en el conjunto de resultados.

Ejemplo:

```
//Desde el controlador

private function buscarDNI($pais,$DNI) {
   if ($fila=$this->modeloContactos->buscarDNI($pais, $DNI)) {
      $this->jInfo("El DNI corresponde a {$fila['nombres']} {$fila['apellido_paterno']}");
   } else {
      $this->jWarning("No se ha encontrado el DNI");
   }
}

// Desde el modelo
public function buscarDNI($pais, $dni) {
   $sql="SELECT nombres, apellido_paterno FROM contactos WHERE pais = :pais AND dni =
:dni"; //La consulta SQL con los parámetros enlazables
   $params=array('pais'=>$pais, 'dni'=>'$dni); //Definimos los valores de los parámetros
   return $this->row($sql, $params); //Retorna la fila que coincida con la consulta
}
```

nextRow($fetchmode): Devuelve la siguiente fila de una consulta aplicada.

rows($query, $params = NULL, $fetchMode = PDO::FETCHASSOC): Es exactamente igual a row(), con la diferencia que puede devolver varias filas que cumplan la condición de la consulta.

Ejemplo:

```
//Desde el controlador

private function obtenerDatos($pais, $ciudad) {
   if ($filas=$this->modeloContactos->obtenerCiudad($pais, $ciudad)) {
      $this->mostrarFilas($filas); //$filas es un arreglo asociativo de contactos
   } else {
      $this->jWarning("Ciudad no encontrada")
      $this->limpiarFilas();
   }
}

// Desde el modelo

public function obtenerCiudad($pais, $ciudad) {
  $sql="SELECT * FROM contactos WHERE país = :país AND ciudad = :ciudad";
  $params=array('pais'=$pais, 'ciudad'=>$ciudad);
  return $this->rows($sql,$params); //retorna todos los contactos del País y ciudad
especificados
}
```

execute($query, $params): Ejecuta consultas de acción (insert, update, replace, delete, truncate) con enlace de parámetros. Devuelve el número de filas afectadas (affected).

Ejemplo:

```
public function eliminar($pais, $ciudad) {
   $sql="DELETE FROM {$this->table} WHERE país=:país AND ciudad = :ciudad";
   $params=array('pais'=>$pais, 'ciudad'=>$ciudad);
   return $this->execute($sql, $params);
}
```

bind($param, $value, $type='none', $trim=false): Enlaza parámetros manualmente.

bindArray($paramarray): Enlaza parámetros manualmente por medio de un arreglo (parámetro, valor).

clearBinding(): Elimina o limpia la asignación de parámetros.

paging($query, $limit=20, $page=1): Devuelve datos paginados de la consulta $query, con un límite de $limit y la página $page, en arreglo asociativo.

Control de Transacciones

El control de transacciones es una metodología para mantener la integridad de los datos, haciendo que estas transacciones no puedan finalizar en un estado intermedio. Cuando por alguna causa el sistema debe cancelar la transacción, empieza a deshacer las órdenes ejecutadas hasta dejar la base de datos en su estado inicial. Esto es posible con bases de datos que soporten la integridad referencial, como es MySQL, PostgreSQL, Oracle, entre otras.

En el caso de MySQL, esta debe tener habilitado el motor de InnoDB, para poder controlar las transacciones.

beginTrans(): comienza el control de transacción.

commitTrans(): termina y confirma la transacción.

rollBack(): Deshace las transacción.

endTrans(): Termina la transacción. Lo que hace es tratar de confirmar la transacción y si hay errores la deshace.

Estos métodos trabajan con la propiedad "trans" que controla si se ha iniciado una transacción.

Ejemplo:

```
public function guardarMaestraDetalle($detalle) {
   if ($this->basicValidation()) {
      return false;
   }
   $this->beginTrans(); // Si no hay errores de validación, comienza el control de transacción
   if ($this->insert()) {
      return $this-guardaDetalles($detalle); // Si se insertó, se solicita guardar el detalle
   } else {
      return false; // Si no devuelve falso
   }
}

private function guardaDetalle($filas) {
   $modeloDetalle = new Model_Detalle(); // Instancia el modelo del detalle
   foreach($filas as $fila) {
      if ($modeloDetalle->insert(false, $fila)==FALSE) {
         $this->rollBack(); // Si hay un error los cambios hechos se deshacen.
         return false; // Y retorna falso
      }
   }
   $this->commitTrans(); // Si no hay errores se confirma la transacción.
   return true;
}
```

CONTROL DE SESIÓN (APJSession)

El módulo de control de sesión permite controlar el acceso de los usuarios al sistema. Para ello cuenta con varios métodos de control.

Métodos:

APJSession::active($name): Nos devuelve si la sesión está aún activa y es la misma que estuvo activa. Si la dirección IP o el navegador cambia, devolverá que no es la misma sesión.

APJSession::start($name, $limit, $path, $domain, $secure): Inicia una sesión. Definiendo un nombre, límite de duración y opcionalmente la ruta de almacenamiento para las cookies, el dominio y se es una conexión segura.

APJSession::destroy(): Termina la sesión. Se utiliza cuando el usuario termina la sesión.

Los valores de sesión son los siguientes:

$_SESSION['id']: Corresponde al ID del usuario que ha iniciado sesión.
$_SESSION['IPAddress']: El número de IP desde donde se inició sesión.
$_SESSION['userAgent']: El navegador desde el cual se inició sesión.
$_SESSION['app']: Arreglo con todos los controladores a los que el usuario tiene acceso.

Al crear un módulo de acceso, debe considerar que debe crear $_SESSION['id'] y $_SESSION['app'], que corresponden al identificador único del usuario y los controladores a los que tiene acceso. Son los únicos que debe asignar, ya que del resto se encarga el control de sesión y el controlador.

Recuerde que desde el controlador al llamar sessionControl() este controlará si se ha iniciado la sesión y si el usuario tiene acceso al controlador actual. Además de asignar la propiedad local userId. Si el usuario no ha iniciado sesión o no tiene acceso al controlador actual, lo redirigirá al LOGIN (definido en init.php)

Para controlar y poblar todos estos datos es recomendable mantener en la base de datos una tabla de usuarios y otra de accesos a los controladores que se encuentre relacionada con los usuarios.