





WIZ - El portal


PLUGINS Y EXTENSIONES

	WIZ – EL PORTAL PLUGINS Y EXTENSIONES	CONSORZIO PISA RICERCHE
		Revisión: 2
		Fecha: 21 junio 2012
Autores: Ing. Salvo Di Mare Tel: +39050931630 E-mail: s.dimare@cpr.it		Página: 2 de 20

	WIZ – EL PORTAL PLUGINS Y EXTENSIONES	CONSORZIO PISA RICERCHE
		Revisión: 2
		Fecha: 21 junio 2012
Autores: Ing. Salvo Di Mare Tel: +39050931630 E-mail: s.dimare@cpr.it		Página: 3 de 20

Índice

Introducción	4
Plugins	5
Version	5
Sql	6
Model	6
Controller	7
Views	7
Extensiones	9
Application Component	9
Behavior	10
Widget	12
Action	14
Filter	15
Controller	16
Validator	16
Helper	17
Module	18


	WIZ – EL PORTAL PLUGINS Y EXTENSIONES	CONSORZIO PISA RICERCHE
		Revisión: 2
		Fecha: 21 junio 2012
Autores: Ing. Salvo Di Mare Tel: +39050931630 E-mail: s.dimare@cpr.it		Página: 4 de 20

Introduzione

Añadir nuevas funciones es una operación muy común durante el ciclo de vida de un software. Para el portal WIZ se han previsto funciones que permiten a un usuario desarrollador, con conocimientos informáticos avanzados, poder ampliar la potencialidad del portal mediante la utilización de plugins.

Además de añadir funciones, también es posible modificar o personalizar aquellas ya existentes; esta operación no se puede realizar utilizando el mecanismo de los plugins sino que es necesario “ampliar” el código, utilizando las funciones que ofrece el marco Yii .

En el primer caso se requiere un conocimiento básico del marco pero es necesario también un buen conocimiento del código fuente del portal; en el segundo, sin embargo, es necesario tener un conocimiento óptimo tanto del marco como de la fuente.

	WIZ – EL PORTAL PLUGINS Y EXTENSIONES	CONSORZIO PISA RICERCHE
		Revisión: 2
		Fecha: 21 junio 2012
Autores: Ing. Salvo Di Mare Tel: +39050931630 E-mail: s.dimare@cpr.it		Página: 5 de 20

Plugins

Un plugin permite añadir nuevas funciones al sistema de manera simple y rápida. Se constituye de un archivo (.tar o .zip), con el nombre que sirve de ID único. A continuación, se muestra la estructura para el plugin Foo:

Foo.tar	plugin archive file
Foo.version	text file containing information about the plugin version
Foo.sql	sql file to create new tables (opzionale)
Foo.php	model class file
FooController.php	controller class file
views/	containing view files

Mediante una simple interfaz gráfica, el usuario desarrollador puede cargar los archivos que constituyen el plugin y habilitar o deshabilitar aquellos ya cargados.

La estructura del archivo debe ser necesariamente la que se ha ilustrado previamente; en caso contrario el sistema no será capaz de instalar el plugin correctamente.

En los párrafos sucesivos, se ilustran las estructuras que contendrán los diferentes files que componen el archivo general.


Version

El file version es un archivo de texto con extensión txt. Está subdividido en secciones, recogido entre corchetes. La sección principal es *version*, obligatoria, y contiene el número de versiones del plugin; las otras dos secciones son opcionales:

- *Description*: contiene una descripción del plugin y de las funciones ofrecidas.
- *Changelog*: contiene información sobre las modificaciones realizadas en la última versión (y si es el caso también en las precedentes)

A continuación se muestra un ejemplo de un hipotético archivo version para el plugin Foo.

[version]

	WIZ – EL PORTAL PLUGINS Y EXTENSIONES	CONSORZIO PISA RICERCHE
		Revisión: 2
		Fecha: 21 junio 2012
Autores: Ing. Salvo Di Mare Tel: +39050931630 E-mail: s.dimare@cpr.it		Página: 6 de 20

1.30-r9 Beta

[description]

A short or detailed description of Foo plugin

[changelog]

The changelog

Sql

En el caso en que el plugin requiera nuevas tablas será necesario incluir en la base de datos un archivo que contenga el sql para crearlas. El portal no tiene en cuenta el contenido del archivo; ejecuta simplemente el código sql. Para realizar esta acción el desarrollador deberá incluir archivos sql válidos y correctos, considerando también el DBMS utilizado. A continuación se muestra un ejemplo de archivo sql.


```
CREATE TABLE foo_table_one
(
  id integer NOT NULL,
  something character varying(255) NOT NULL,
  CONSTRAINT id_pkey PRIMARY KEY(id)
)
```

Model

El modelo es una clase que debe ampliar CFormModel o CActiveRecord. Se utiliza para manipular y memorizar los datos. Representa un único concepto que puede ser una linea en una tabla o un formulario html con campos de input. Cada campo del concepto se representa por un atributo del modelo.

El nombre de la clase debe ser igual al del archivo del modelo (excepto la extensión .php).

```
class Foo extends CActiveRecord
{
  public static function model($className=__CLASS__)
  {
    return parent::model($className);
  }
}
```

	WIZ – EL PORTAL PLUGINS Y EXTENSIONES	CONSORZIO PISA RICERCHE
		Revisión: 2
		Fecha: 21 junio 2012
Autores: Ing. Salvo Di Mare Tel: +39050931630 E-mail: s.dimare@cpr.it		Página: 7 de 20

```

public function tableName()
{
    return 'foo_table_one';
}

```

Controller

El controller debe ampliar la clase CExtController o cualquier otro controller ya presente en el portal. El controller ejecuta las diferentes acciones requeridas, utilizando si fuese necesario los datos del modelo y visualizando la información mediante las view.

```

class FooController extends CExtController
{
    public function actionIndex()
    {
        // ...
    }
    // other actions
}

```

Views

Todas las view deben estar organizadas dentro de la carpeta views. Una view no es más que un archivo html que contendrá generalmente el código en php.

```

<div class="top">
    <h1>Title</h1>
    <?php echo $content; ?>
</div>

```


Las view son utilizadas, dentro del controller, por las diferentes action para mostrar datos e información al usuario. El ejemplo siguiente muestra cómo invocar la view edit de una action cualquiera de FooController.

```

$this->render('edit', array(
    'var1'=>$value1,
    'var2'=>$value2,
));

```


A continuación, se visualiza cómo invocar una view ya existente y relativa al componente bar.

	WIZ – EL PORTAL PLUGINS Y EXTENSIONES	CONSORZIO PISA RICERCHE
		Revisión: 2
		Fecha: 21 junio 2012
Autores: Ing. Salvo Di Mare Tel: +39050931630 E-mail: s.dimare@cpr.it		Página: 8 de 20

```

$this->render('//bar/view', array(
    'var1'=>$value1,
    'var2'=>$value2,
));

```


	WIZ – EL PORTAL PLUGINS Y EXTENSIONES	CONSORZIO PISA RICERCHE
		Revisión: 2
		Fecha: 21 junio 2012
Autores: Ing. Salvo Di Mare Tel: +39050931630 E-mail: s.dimare@cpr.it		Página: 9 de 20

Extensiones

Una extensión sirve normalmente para un objetivo concreto. Las extensiones pueden clasificarse en:

- application component
- behavior
- widget
- action
- filter
- controller
- validator
- helper
- module


La extensión puede ser también un componente que no entra en ninguna de las categorías mencionadas. Yii está estudiado de tal modo que casi cada parte de su código puede extenderse o personalizarse para adaptarse a las exigencias individuales.

Application Component

Una application component debe aplicar la interfaz `IApplicationComponent` o extenderse desde `CapplicationComponent`. El método principal que se debe aplicar es `IApplicationComponent::init`, gracias a él el componente ejecuta las operaciones de inicialización. Este método se ejecuta inmediatamente después de que el componente se haya creado y establezca los valores iniciales de la propiedad (especificadas en la configuración de la aplicación).

Por defecto, un application component se crea e inicializa solamente cuando se accede por primera vez durante la gestión de una solicitud. Además, si la application component se debe crear inmediatamente después de la creación de una solicitud de la aplicación, el usuario deberá introducir el ID del componente en la propiedad `Capplication::preload`.

Para utilizar un application componente es necesario sobre todo modificar el archivo de configuración de la aplicación añadiendo una nueva property, como se muestra a continuación:

	WIZ – EL PORTAL PLUGINS Y EXTENSIONES	CONSORZIO PISA RICERCHE
		Revisión: 2
		Fecha: 21 junio 2012
Autores: Ing. Salvo Di Mare Tel: +39050931630 E-mail: s.dimare@cpr.it		Página: 10 de 20

```

return array(
    // 'preload'=>array('xyz',...),
    'components'=>array(
        'xyz'=>array(
            'class'=>'ext.xyz.XyzClass',
            'property1'=>'value1',
            'property2'=>'value2',
        ),
        // other component configurations
    ),
);

```

Después se puede acceder al componente en cualquier parte del código utilizando `Yii::app()->xyz`. El componente no se creará inmediatamente a continuación de acceder la primera vez a menos que no se haya introducido en la propiedad `preload`.

Behavior

Para crear un behavior es necesario aplicar la interfaz `IBehavior`. Para su comodidad, Yii incluye la clase `Cbehavior` que aplica esta interfaz y proporciona cómodos métodos añadidos. Las clases derivadas deberán principalmente aplicar los métodos extra que ponen a disposición de los componentes a los que están conectados.


Cuando se desarrolla un behavior para `CModel` y `CActiveRecord`, se puede extender desde `CModelBehavior` y `CActiveRecordBehavior` respectivamente. Estas clases ofrecen funciones adicionales específicas para `CModel` y `CActiveRecord`.

El siguiente código muestra un ejemplo de un `ActiveRecord` behavior. Cuando este behavior se conecta a un concepto AR, en cuanto el concepto se guarda con el método `save()`, este configura automáticamente el valor del atributo `create_time` e `update_time` con el timestamp actual.

```

class TimestampBehavior extends CActiveRecordBehavior
{
    public function beforeSave($event)
    {
        if($this->owner->isNewRecord)
            $this->owner->create_time=time();
    }
}

```

	WIZ – EL PORTAL PLUGINS Y EXTENSIONES	CONSORZIO PISA RICERCHE
		Revisión: 2
		Fecha: 21 junio 2012
Autores: Ing. Salvo Di Mare Tel: +39050931630 E-mail: s.dimare@cpr.it		Página: 11 de 20

```

else
    $this->owner->update_time=time();
}
}

```

Un behavior puede utilizarse en cualquier componente. La utilización requiere dos pasos: conectar el behavior al componente e invocar el behavior. Por ejemplo:

```

// $name uniquely identifies the behavior in the component
$component->attachBehavior($name,$behavior);
// test() is a method of $behavior
$component->test();

```

A menudo, un behavior se conecta a un componente utilizando el archivo de configuración en lugar de invocar el método attachBehavior. Por ejemplo:

```

return array(
    'components'=>array(
        'db'=>array(
            'class'=>'CDbConnection',
            'behaviors'=>array(
                'xyz'=>array(
                    'class'=>'ext.xyz.XyzBehavior',
                    'property1'=>'value1',
                    'property2'=>'value2',
                ),
            ),
        ),
        //....
    ),
);

```

Este código conecta el behavior xyz a la application component db.

Para las clases CController, CFormModel y CActiveRecord que generalmente se extienden, los behavior pueden conectarse realizando el override del método behaviors(). Por ejemplo:



WIZ – EL PORTAL PLUGINS Y EXTENSIONES

CONSORZIO PISA RICERCHE

Revisión: 2

Fecha: 21 junio 2012

Autores: Ing. Salvo Di Mare
Tel: +39050931630
E-mail: s.dimare@cpr.it

Página: 12 de 20

```
public function behaviors()  
{  
    return array(  
        'xyz'=>array(  
            'class'=>'ext.xyz.XyzBehavior',  
            'property1'=>'value1',  
            'property2'=>'value2',  
        ),  
    );  
}
```


Widget

Un widget extiende CWidget o una clase derivada.

El método más simple para crear un nuevo widget es el de extender un widget existente y realizar el override de sus métodos o modificar los valores por defecto de las propiedades. Por ejemplo, si se quiere utilizar un estilo CSS diferente para CtabView se puede configurar la propiedad CtabView::cssfile cuando se utiliza el widget o se puede extender CtabView como se muestra a continuación para no tener que configurar esta propiedad cada vez que se utilice.

```
class MyTabView extends CTabView  
{  
    public function init()  
    {  
        if($this->cssFile===null)  
        {  
            $file=dirname(__FILE__).DIRECTORY_SEPARATOR.'tabview.css';  
            $this->cssFile=Yii::app()->getAssetManager()->publish($file);  
        }  
        parent::init();  
    }  
}
```

En el código que se muestra arriba se ha realizado el override del método Cwidget::init y asignado a CtabView::cssFile el URL del nuevo estilo CSS, si la propiedad no se ha configurado previamente. El nuevo estilo CSS se coloca en el mismo directorio que contiene la clase MyTabView, de manera que todo se pueda

	WIZ – EL PORTAL PLUGINS Y EXTENSIONES	CONSORZIO PISA RICERCHE
		Revisión: 2
		Fecha: 21 junio 2012
Autores: Ing. Salvo Di Mare Tel: +39050931630 E-mail: s.dimare@cpr.it		Página: 13 de 20

dejar como extensión. Además no se puede acceder al archivo CSS desde la pagina web, sino que hay que publicarlo.

Para crear un nuevo widget desde cero, es necesario aplicar dos métodos: CWidget::init y CWidget::run. El primer método se aplica cuando se utiliza \$this->beginWidget para introducir el widget en una vista mientras que el segundo se invoca cuando se utiliza \$this->endWidget. Si se quiere capturar y procesar el contenido mostrado entre la aplicación de estos dos métodos es necesario guardar en la memoria el output en CWidget::init para poder recuperarlo después en CWidget::run en caso de un procesamiento.

Un widget requiere a menudo CSS, Javascript y otros archivos en la página que utiliza el propio widget. Estos archivos normalmente se llaman assets porque van junto al archivo que contiene el widget y los usuarios pueden acceder a ellos desde la web. Para hacer que estos archivos estén disponibles en la web es necesario publicarlos utilizando CWebApplication::assetManager, como se muestra en el fragmento del código indicado a continuación. Además, si fuese necesario incluir un archivo CSS o un Javascript en la página se deberá registrar utilizando CClientScript.

```
class MyWidget extends CWidget
{
    protected function registerClientScript()
    {
        // ...publish CSS or JavaScript file here...
        $cs=Yii::app()->clientScript;
        $cs->registerCssFile($cssFile);
        $cs->registerScriptFile($jsFile);
    }
}
```

Un widget puede también incluir los propios archivos para las vistas. En este caso será necesario crear un directorio llamado views bajo el directorio que contiene la clase del widget y posicionar todas las vistas dentro de él. En una clase widget, para mostrar una vista, se utiliza \$this->render('ViewName').

Dada una widget XyzClass que pertenece a la extensión xyz, podemos volver a renombrarla en una view como muestra el siguiente código:

```
// widget that does not need body content
<?php $this->widget('ext.xyz.XyzClass', array(
    'property1'=>'value1',
    'property2'=>'value2')); ?>
```



WIZ – EL PORTAL PLUGINS Y EXTENSIONES

CONSORZIO PISA RICERCHE

Revisión: 2

Fecha: 21 junio 2012

Autores: Ing. Salvo Di Mare
Tel: +39050931630
E-mail: s.dimare@cpr.it

Página: 14 de 20

```
// widget that can contain body content
<?php $this->beginWidget('ext.xyz.XyzClass', array(
    'property1'=>'value1',
    'property2'=>'value2')); ?>

...body content of the widget...

<?php $this->endWidget(); ?>
```


Action

Una action extiende la clase CAction o una clase derivada. El método principal que se debe aplicar es IAction::run.

Las action se utilizan en un controller para responder a los requisitos específicos. Dada una action XyzClass que pertenece a la extensión xyz, la invocación puede realizarse haciendo el override del método CController::action de la clase controller:

```
class TestController extends CController
{
    public function actions()
    {
        return array(
            'xyz'=>array(
                'class'=>'ext.xyz.XyzClass',
                'property1'=>'value1',
                'property2'=>'value2',
            ),
            // other actions
        );
    }
}
```

Se podrá acceder a la action siguiendo test/xyz.

	WIZ – EL PORTAL PLUGINS Y EXTENSIONES	CONSORZIO PISA RICERCHE
		Revisión: 2
		Fecha: 21 junio 2012
Autores: Ing. Salvo Di Mare Tel: +39050931630 E-mail: s.dimare@cpr.it		Página: 15 de 20

Filter

Un filter extiende la clase CFilter o una clase derivada. Los métodos principales que se deben aplicar son CFilter::preFilter e CFilter::postFilter. El primero se invoca antes de que la action sea ejecutada mientras que el segundo después.


```
class MyFilter extends CFilter
{
    protected function preFilter($filterChain)
    {
        // Logic being applied before the action is executed
        return true; // false if the action should not be executed
    }

    protected function postFilter($filterChain)
    {
        // Logic being applied after the action is executed
    }
}
```

El parámetro \$filterChain es de tipo CFilterChain y contiene información sobre la action actual.

Dado un filter XyzClass que pertenece a la extensión xyz, este se podrá utilizar haciendo el override del método CController::filters de la clase controller:

```
class TestController extends CController
{
    public function filters()
    {
        return array(
            array(
                'ext.xyz.XyzClass',
                'property1'=>'value1',
                'property2'=>'value2',
            ),
            // other filters
        );
    }
}
```

	WIZ – EL PORTAL PLUGINS Y EXTENSIONES	CONSORZIO PISA RICERCHE
		Revisión: 2
		Fecha: 21 junio 2012
Autores: Ing. Salvo Di Mare Tel: +39050931630 E-mail: s.dimare@cpr.it		Página: 16 de 20

```
}
```

Controller

Un controller distribuido como extensión debería extender CExtController, en lugar de CController. El motivo principal es que CController prevé que los archivos de las vistas se encuentren en `application.views.ControllerID` mientras que CExtController prevé que los archivos se encuentren en el directorio `views`, que es un subdirectorio del que contiene la clase controller.

Un controller proporciona un conjunto de actions que puede solicitar el usuario. Para poder utilizar una extensión controller es necesario configurar la propiedad `CWebApplication::controllerMap` al final de la configuración de la aplicación:

```
return array(
    'controllerMap'=>array(
        'xyz'=>array(
            'class'=>'ext.xyz.XyzClass',
            'property1'=>'value1',
            'property2'=>'value2',
        ),
        // other controllers
    ),
);
```

Se puede acceder a la action `a` del controller mediante `xyz/a`.

Validator

Un validator se debe extender desde CValidator y aplicar el método `CValidator::validateAttribute`

```
class MyValidator extends CValidator
{
    protected function validateAttribute($model,$attribute)
    {
```




WIZ – EL PORTAL PLUGINS Y EXTENSIONES

CONSORZIO PISA RICERCHE

Revisión: 2

Fecha: 21 junio 2012

Autores: Ing. Salvo Di Mare
Tel: +39050931630
E-mail: s.dimare@cpr.it

Página: 17 de 20

```
$value=$model->$attribute;  
if($value has error)  
    $model->addError($attribute,$errorMessage);  
}  
}
```


Un validator se utiliza principalmente en una clase model (o en cualquiera que extiende CFormModel o CActiveRecord). Dado un validator `XYZClass` que pertenece a la extensión `xyz`, este se puede utilizar realizando el override del método `CModel::run` de la clase model:

```
class MyModel extends CActiveRecord // or CFormModel  
{  
    public function rules()  
    {  
        return array(  
            array(  
                'attr1, attr2',  
                'ext.xyz.XyzClass',  
                'property1'=>'value1',  
                'property2'=>'value2',  
            ),  
            // other validation rules  
        );  
    }  
}
```

Helper

Es una clase formada solamente de metodos estadísticos. Su comportamiento es parecido al de las funciones globales; para invocarlo se utiliza el nombre de la clase como namespace.

```
class MyHelper  
{  
    public static function utility($param)  
    {  
        // ...  
    }  
}
```

	WIZ – EL PORTAL PLUGINS Y EXTENSIONES	CONSORZIO PISA RICERCHE
		Revisión: 2
		Fecha: 21 junio 2012
Autores: Ing. Salvo Di Mare Tel: +39050931630 E-mail: s.dimare@cpr.it		Página: 18 de 20

```
// ...code here...
}
}
```

```
// .....
MyHelper::utility($value)
// .....
```

Module

Un module extiende la clase CWebModule; consiste en models, views y controllers.


Una directriz general para el desarrollo de un module prevé que sea self-contained. Los archivos (como los CSS, los Javascript y las imágenes) que se utilizan desde un módulo deben distribuirse junto a tal module. El module tiene que publicar estos archivos de modo que se pueda acceder a ellos desde la web.

Un module está organizado en un directorio, con el nombre que actúa como ID único. A continuación se muestra una estructura de directorio típica del module forum:

```
forum/

  ForumModule.php          the module class file

  components/              containing reusable user components
    views/                 containing view files for widgets
  controllers/             containing controller class files
    DefaultController.php  the default controller class file
  extensions/              containing third-party extensions
  models/                  containing model class files
  views/                   containing controller view and layout files
    layouts/               containing layout view files
```

	WIZ – EL PORTAL PLUGINS Y EXTENSIONES	CONSORZIO PISA RICERCHE
		Revisión: 2
		Fecha: 21 junio 2012
Autores: Ing. Salvo Di Mare Tel: +39050931630 E-mail: s.dimare@cpr.it		Página: 19 de 20

default/	containing view files for DefaultController
index.php	the index view file

Para utilizar un module, es necesario sobre todo copiar el module en el directorio modules de la aplicación. A continuación es necesario añadir el ID del module en la propiedad modules de la aplicación. Por ejemplo, para poder utilizar el module forum, es necesario utilizar la siguiente configuración:

```
return array(
    .....
    'modules'=>array('forum',...),
    .....
);
```


Un module puede también tener propiedades que se tengan que inicializar. Por ejemplo, el module forum puede tener la propiedad postPerPage que se puede configurar del siguiente modo:

```
return array(
    .....
    'modules'=>array(
        'forum'=>array(
            'postPerPage'=>20,
        ),
    ),
    .....
);
```

Se puede acceder a la solicitud de un module mediante la propiedad module del controller activo. Mediante la solicitud es posible acceder a la información compartida a nivel de module. Por ejemplo, para acceder a la propiedad postPerPage se puede utilizar:

```
$postPerPage=Yii::app()->controller->module->postPerPage;
// or the following if $this refers to the controller instance
// $postPerPage=$this->module->postPerPage;
```

Se puede acceder a la action de un controller que se encuentra dentro de un module mediante moduleID/controllerID/actionID.

	WIZ – EL PORTAL PLUGINS Y EXTENSIONES	CONSORZIO PISA RICERCHE
		Revisión: 2
		Fecha: 21 junio 2012
Autores: Ing. Salvo Di Mare Tel: +39050931630 E-mail: s.dimare@cpr.it		Página: 20 de 20

Por ejemplo, suponiendo que el module forum tiene un controller con nombre PostController, se podrá acceder a la action create del controller mediante forum/post/create. El url correspondiente será <http://www.example.com/index.php?r=forum/post/create>.