





WIZ - Il portale


PLUGINS ED ESTENSIONI

	WIZ – IL PORTALE PLUGINS ED ESTENSIONI	CONSORZIO PISA RICERCHE
		Revisione: 2
		Data: 21 Giugno 2012
Autori: Ing. Salvo Di Mare Tel: +39050931630 E-mail: s.dimare@cpr.it		Pagina: 2 di 20

	WIZ – IL PORTALE PLUGINS ED ESTENSIONI	CONSORZIO PISA RICERCHE
		Revisione: 2
		Data: 21 Giugno 2012
Autori: Ing. Salvo Di Mare Tel: +39050931630 E-mail: s.dimare@cpr.it		Pagina: 3 di 20

Sommario

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


	WIZ – IL PORTALE PLUGINS ED ESTENSIONI	CONSORZIO PISA RICERCHE
		Revisione: 2
		Data: 21 Giugno 2012
Autori: Ing. Salvo Di Mare Tel: +39050931630 E-mail: s.dimare@cpr.it		Pagina: 4 di 20

Introduzione

Aggiungere nuove funzionalità è un'operazione molto comune durante il ciclo di vita di un software. Per il portale WIZ sono state previste delle funzionalità per permettere ad un utente sviluppatore, quindi con conoscenze informatiche avanzate, di poter ampliare le potenzialità del portale stesso tramite l'utilizzo di plugins.

Oltre che aggiungere funzionalità è anche possibile modificare o, in generale, personalizzare quelle esistenti; questa operazione non può essere effettuata utilizzando il meccanismo dei plugin ma bisogna 'estendere' il codice, utilizzando le potenzialità offerte dal framework Yii .

Nel primo caso è richiesta una conoscenza marginale del framework ma una buona conoscenza del codice sorgente del portale; nel secondo, invece, è necessario avere ottime conoscenze sia del framework che del sorgente.

	WIZ – IL PORTALE PLUGINS ED ESTENSIONI	CONSORZIO PISA RICERCHE
		Revisione: 2
		Data: 21 Giugno 2012
Autori: Ing. Salvo Di Mare Tel: +39050931630 E-mail: s.dimare@cpr.it		Pagina: 5 di 20

Plugins

Un plugin permette di aggiungere nuove funzionalità al sistema in maniera semplice e veloce. È costituito da un archivio (.tar o .zip), con il nome che funge da ID univoco. Di seguito è mostrata la struttura per il 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

Attraverso una semplice interfaccia grafica l'utente sviluppatore può caricare gli archivi che costituiscono il plugin ed abilitare o disabilitare quelli preventivamente caricati.

La struttura dell'archivio deve necessariamente essere quella illustrata sopra altrimenti il sistema non sarà in grado di installare il plugin correttamente.

Nei paragrafi successivi verranno illustrate le strutture che dovranno avere i vari file che compongono l'archivio.


Version

Il file version è un file di testo con estensione txt. È suddiviso in sezioni, racchiuse da parentesi quadre. La sezione principale è *version*, obbligatoria, e contiene il numero di versione del plugin; le altre due sezioni sono opzionali:

- *Description*: contiene una descrizione del plugin e delle funzionalità offerte
- *Changelog*: contiene informazioni sulle modifiche apportate nell'ultima versione (ed eventualmente nelle precedenti).

Di seguito è mostrato un esempio di un ipotetico file version per il plugin Foo.

[version]

	WIZ – IL PORTALE PLUGINS ED ESTENSIONI	CONSORZIO PISA RICERCHE
		Revisione: 2
		Data: 21 Giugno 2012
Autori: Ing. Salvo Di Mare Tel: +39050931630 E-mail: s.dimare@cpr.it		Pagina: 6 di 20

1.30-r9 Beta

[description]

A short or detailed description of Foo plugin

[changelog]

The changelog

Sql

Nel caso in cui il plugin necessiti di nuove tabelle nel database è necessario fornire un file contenente l'sql per creare tali tabelle. Il portale non entra nel merito del contenuto del file; esegue semplicemente il codice sql per cui è responsabilità dello sviluppatore fornire file sql validi e formalmente corretti, considerando anche il DBMS utilizzato. Di seguito è mostrato un esempio di file sql.


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

Model

Il modello è una classe che deve estendere CFormModel o CActiveRecord. Viene utilizzato per manipolare e memorizzare i dati. Rappresenta un singolo oggetto che può essere una riga in una tabella o una form html con dei campi di input. Ogni campo dell'oggetto è rappresentato da un attributo del modello.

Il nome della classe deve essere uguale a quello del file del modello (esclusa l'estensione .php)

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

	WIZ – IL PORTALE PLUGINS ED ESTENSIONI	CONSORZIO PISA RICERCHE
		Revisione: 2
		Data: 21 Giugno 2012
Autori: Ing. Salvo Di Mare Tel: +39050931630 E-mail: s.dimare@cpr.it		Pagina: 7 di 20

```

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

```

Controller

Il controller deve estendere la classe CExtController o qualsiasi altro controller già presente nel portale. Il controller esegue le varie action richieste, utilizzando eventualmente i dati del modello e visualizzando le informazioni tramite le view.

```

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

```

Views

Tutte le view devono essere organizzate dentro la cartella views. Una view altro non è che un file html, con dentro eventualmente del codice in php.

```

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

```


Le view vengono utilizzate, all'interno del controller, dalle varie action per visualizzare dati e informazioni all'utente. L'esempio seguente mostra come invocare la view edit da una qualsiasi action di FooController.

```

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

```


Di seguito, invece, viene mostrato come invocare una view già esistente e relativa al componente bar.

	WIZ – IL PORTALE PLUGINS ED ESTENSIONI	CONSORZIO PISA RICERCHE
		Revisione: 2
		Data: 21 Giugno 2012
Autori: Ing. Salvo Di Mare Tel: +39050931630 E-mail: s.dimare@cpr.it		Pagina: 8 di 20

```

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

```


	WIZ – IL PORTALE PLUGINS ED ESTENSIONI	CONSORZIO PISA RICERCHE
		Revisione: 2
		Data: 21 Giugno 2012
Autori: Ing. Salvo Di Mare Tel: +39050931630 E-mail: s.dimare@cpr.it		Pagina: 9 di 20

Estensioni

Una estensione tipicamente serve per uno scopo ben preciso. Le estensioni possono essere classificate in:

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


L'estensione può essere anche un componente che non rientra in nessuna delle categorie di cui sopra. Yii è studiato in modo tale che quasi ogni pezzo del suo codice può essere esteso e personalizzato per adattarsi alle esigenze individuali.

Application Component

Un application component deve implementare l'interfaccia `IApplicationComponent` o estendere da `CApplicationComponent`. Il metodo principale che deve essere implementato è `IApplicationComponent::init`, grazie al quale il componente esegue le operazioni di inizializzazione. Questo metodo viene invocato subito dopo che il componente è stato creato e setta i valori iniziali delle proprietà (specificate nella configurazione dell'applicazione).

Per default, un application component viene creato ed inizializzato solo quando viene acceduto per la prima volta durante la gestione di una richiesta. Tuttavia, se l'application component deve essere creato subito dopo la creazione di una istanza dell'applicazione, è compito dell'utente inserire l'ID del componente nella proprietà `CApplication::preload`.

Per utilizzare un application component bisogna innanzitutto modificare il file di configurazione dell'applicazione aggiungendo una nuova property, come mostra di seguito:

	WIZ – IL PORTALE PLUGINS ED ESTENSIONI	CONSORZIO PISA RICERCHE
		Revisione: 2
		Data: 21 Giugno 2012
Autori: Ing. Salvo Di Mare Tel: +39050931630 E-mail: s.dimare@cpr.it		Pagina: 10 di 20

```

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

```

Successivamente si può accedere al componente in qualsiasi punto del codice utilizzando `Yii::app()->xyz`. Il componente sarà creato non appena verrà acceduto per la prima volta a meno che non sia stato inserito nella proprietà `preload`.

Behavior

Per creare un behavior bisogna implementare l'interfaccia `IBehavior`. Per comodità, Yii fornisce la classe `CBehavior` che implementa già questa interfaccia e fornisce comodi metodi aggiuntivi. Le classi figlie dovranno principalmente implementare i metodi extra che intendono mettere a disposizione dei componenti ai quali sono collegati.


Quando si sviluppa un behavior per `CModel` e `CActiveRecord`, si può anche estendere da `CModelBehavior` e `CActiveRecordBehavior`, rispettivamente. Queste classi offrono funzionalità aggizionali specifiche per `CModel` e `CActiveRecord`.

Il codice seguente mostra un esempio di un `ActiveRecord` behavior. Quando questo behavior è collegato ad un oggetto AR, non appena l'oggetto viene salvato chiamando il metodo `save()`, esso setta automaticamente il valore dell'attributo `create_time` e `update_time` con il timestamp corrente.

```

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

```

	WIZ – IL PORTALE PLUGINS ED ESTENSIONI	CONSORZIO PISA RICERCHE
		Revisione: 2
		Data: 21 Giugno 2012
Autori: Ing. Salvo Di Mare Tel: +39050931630 E-mail: s.dimare@cpr.it		Pagina: 11 di 20

```

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

```

Un behavior può essere utilizzato in qualsiasi componente. L'utilizzo richiede due step: collegare il behavior al componente ed invocare il behavior. Per esempio:

```

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

```

Spesso, un behavior viene collegato ad un componente utilizzando il file di configurazione invece che invocare il metodo attachBehavior. Per esempio:


```

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

```

Questo codice collega il behavior xyz all'applicazione component db.

Per le classi CController, CFormModel e CActiveRecord che generalmente vengono estese, i behavior possono essere collegati effettuando l'override del metodo behaviors(). Per esempio:

	WIZ – IL PORTALE PLUGINS ED ESTENSIONI	CONSORZIO PISA RICERCHE
		Revisione: 2
		Data: 21 Giugno 2012
Autori: Ing. Salvo Di Mare Tel: +39050931630 E-mail: s.dimare@cpr.it		Pagina: 12 di 20

```

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

```

Widget

Un widget estende CWidget o una sua classe figlia.


Il metodo più semplice per creare un nuovo widget è quello di estendere un widget esistente ed effettuare l'override dei suoi metodi o modificare i valori di default delle proprietà. Per esempio, se si vuole utilizzare uno stile CSS più carino per CTabView si può configurare la proprietà CTabView::cssfile quando si usa il widget oppure si può estendere CTabView come mostrato sotto in modo da non dover più configurare questa proprietà ad ogni utilizzo.

```

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();
    }
}

```

Nel codice mostrato sopra viene effettuato l'override del metodo CWidget::init ed assegnato a CTabView::cssFile l'URL del nuovo stile CSS, se la proprietà non è già stata settata. Il nuovo stile CSS viene

	WIZ – IL PORTALE PLUGINS ED ESTENSIONI	CONSORZIO PISA RICERCHE
		Revisione: 2
		Data: 21 Giugno 2012
Autori: Ing. Salvo Di Mare Tel: +39050931630 E-mail: s.dimare@cpr.it		Pagina: 13 di 20

messo nella stessa directory che contiene la classe MyTabView, in modo che il tutto possa anche essere rilasciato come estensione. Poiché il file CSS non è accessibile da Web, è necessario pubblicarlo.

Per creare un nuovo widget da zero, è necessario implementare due metodi: CWidget::init e CWidget::run. Il primo metodo viene invocato quando si utilizza \$this->beginWidget per inserire il widget in una vista, mentre il secondo è invocato quando si utilizza \$this->endWidget. Se si vuole catturare e processare il contenuto mostrato tra l'invocazione di questi due metodi bisogna bufferizzare l'output in CWidget::init per poi recuperarlo in CWidget::run per eventuali processamenti.

Un widget richiede spesso CSS, Javascript e altri file nella pagina che utilizza il widget stesso. Questi file vengono generalmente chiamati assets perché stanno insieme al file che contiene il widget e non sono accessibili dagli utenti web. Per rendere questi file accessibili da web è necessario pubblicarli utilizzando CWebApplication::assetManager, come mostrato nel frammento di codice sottostante. Inoltre, se bisogna includere un file CSS o un Javascript nella pagina è necessario registrarlo utilizzando 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 può anche i propri file per le viste. In questo caso bisogna creare una directory chiama views sotto la directory contenente la classe del widget e posizionare tutte le viste dentro questa directory. In una classe widget, per mostrare una vista, si utilizza \$this->render('ViewName').

Data una widget XyzClass appartenente all'estensione xyz, possiamo richiamarla in una view come mostra il codice seguente:

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

// widget that can contain body content
```

	WIZ – IL PORTALE PLUGINS ED ESTENSIONI	CONSORZIO PISA RICERCHE
		Revisione: 2
		Data: 21 Giugno 2012
Autori: Ing. Salvo Di Mare Tel: +39050931630 E-mail: s.dimare@cpr.it		Pagina: 14 di 20

```
<?php $this->beginWidget('ext.xyz.XyzClass', array(
    'property1'=>'value1',
    'property2'=>'value2')); ?>
```

...body content of the widget...

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


Action

Una action estende la classe CAction o una sua classe figlia. Il metodo principale che deve essere implementato è IAction::run.

Le action vengono utilizzate in un controller per rispondere a specifiche richieste. Data una action XyzClass appartenente all'estensione xyz, l'invocazione può essere effettuata facendo l'override del metodo CController::action della classe controller:

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

La action sarà accessibile tramite il percorso test/xyz.

	WIZ – IL PORTALE PLUGINS ED ESTENSIONI	CONSORZIO PISA RICERCHE
		Revisione: 2
		Data: 21 Giugno 2012
Autori: Ing. Salvo Di Mare Tel: +39050931630 E-mail: s.dimare@cpr.it		Pagina: 15 di 20

Filter

Un filter estende la classe CFilter o una sua classe figlia. I metodi principali che devono essere implementati sono CFilter::preFilter e CFilter::postFilter. Il primo viene invocato prima che la action venga eseguita mentre il secondo dopo.


```
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
    }
}
```

Il parametro \$filterChain è di tipo CFilterChain e contiene informazioni sulla action corrente.

Data un filter XyzClass appartenente all'estensione xyz, esso può essere utilizzato facendo l'override del metodo CController::filters della classe controller:

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

	WIZ – IL PORTALE PLUGINS ED ESTENSIONI	CONSORZIO PISA RICERCHE
		Revisione: 2
		Data: 21 Giugno 2012
Autori: Ing. Salvo Di Mare Tel: +39050931630 E-mail: s.dimare@cpr.it		Pagina: 16 di 20

```
}
```

Controller

Un controller distribuito come estensione dovrebbe estendere CExtController, invece che CController. Il motivo principale è che CController assume che i file delle viste risiedano sotto application.views.ControllerID mentre CExtController assume che i file si trovino nella directory views che è una sottodirectory di quella che contiene la classe controller.

Un controller fornisce un insieme di actions che possono essere richieste dall'utente. Al fine di poter utilizzare una estensione controller è necessario configurare la proprietà CWebApplication::controllerMap nel file di configurazione dell'applicazione:

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

La action a del controller può essere acceduta tramite il percorso xyz/a.

Validator

Un validator deve estendere da CValidator ed implementare il metodo CValidator::validateAttribute

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




WIZ – IL PORTALE PLUGINS ED ESTENSIONI

CONSORZIO PISA RICERCHE

Revisione: 2

Data: 21 Giugno 2012

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

Pagina: 17 di 20

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


Un validator è principalmente utilizzato in una classe model (o una qualsiasi che estende CFormModel o CActiveRecord). Data un validator `XYZClass` appartenente all'estensione `xyz`, esso può essere utilizzato effettuando l'override del metodo `CModel::run` della classe 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

È una classe formata solo da metodi statici. Il suo comportamento è simile a quello delle funzioni globali; per invocarle si utilizza il nome della classe come namespace.

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

	WIZ – IL PORTALE PLUGINS ED ESTENSIONI	CONSORZIO PISA RICERCHE
		Revisione: 2
		Data: 21 Giugno 2012
Autori: Ing. Salvo Di Mare Tel: +39050931630 E-mail: s.dimare@cpr.it		Pagina: 18 di 20

```

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

```

```

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

```

Module

Un module estende la classe CWebModule; consiste di models, views and controllers.

Una linea guida generale per lo sviluppo di un module prevede che esso sia self-contained. I file (come i CSS, i Javascript e le immagini) che sono utilizzati da un module devono essere distribuiti insieme al module stesso. Il module deve anche pubblicare questi file in modo da renderli accessibili da web.

Un module è organizzato in una directory, con il nome che funge da ID univoco. Di seguito è mostrata una struttura di directory tipica 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 – IL PORTALE PLUGINS ED ESTENSIONI	CONSORZIO PISA RICERCHE
		Revisione: 2
		Data: 21 Giugno 2012
Autori: Ing. Salvo Di Mare Tel: +39050931630 E-mail: s.dimare@cpr.it		Pagina: 19 di 20

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

Per utilizzare un module, bisogna innanzitutto copiare il module nella directory `modules` dell'applicazione. Successivamente bisogna dichiarare aggiungere l'ID del module nella proprietà `modules` dell'applicazione. Per esempio, per poter utilizzare il module `forum`, bisogna utilizzare la seguente configurazione:

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


Un module può anche avere delle proprietà che devono essere inizializzate. Per esempio, il module `forum` può avere la proprietà `postPerPage` che può essere configurata nel seguente modo:

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

L'istanza di un module può essere acceduta tramite la proprietà `module` del controller attivo. Attraverso l'istanza è possibile accedere alle informazioni che sono condivise a livello di module. Per esempio, per accedere alla proprietà `postPerPage` si può utilizzare:

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

La action di un controller che si trova dentro un module può essere acceduta tramite il percorso `moduleID/controllerID/actionID`.

	WIZ – IL PORTALE PLUGINS ED ESTENSIONI	CONSORZIO PISA RICERCHE
		Revisione: 2
		Data: 21 Giugno 2012
Autori: Ing. Salvo Di Mare Tel: +39050931630 E-mail: s.dimare@cpr.it		Pagina: 20 di 20

Per esempio, ipotizzando che il module forum abbia un controller di nome PostController, è possibile accedere alla action create del controller tramite il percorso forum/post/create. L'url corrispondente sarà <http://www.example.com/index.php?r=forum/post/create>.